On the Leakage Resilience of Secure Channel Establishment

by

Janaka Araliya Bandara Alawatugoda

BSc (Hons) specializing Computer Science *(University of Peradeniya)* – September 2011

Thesis submitted in accordance with the regulations for the Degree of Doctor of Philosophy

School of Electrical Engineering and Computer Science
Science and Engineering Faculty
Queensland University of Technology
Australia

November 2015

Keywords

Leakage-resilient, Side-channel attacks, Authenticated key exchange, Secure channel, Public-key cryptography, Data compression, CRIME, BREACH

Abstract

Typically, secure channels are constructed from an authenticated key exchange (AKE) protocol, which authenticates the communicating parties based on long-term public keys and establishes secret session keys, and a secure data transmission layer which uses the secret session keys to encrypt transmitting data. In this research we address the partial leakage of long-term secret keys of key exchange protocol participants due to various side-channel attacks, and the partial leakage of plaintexts due to data compression prior to the encryption. Both issues can negatively affect the security of channel establishment and data transmission.

Security models for two-party authenticated key exchange protocols have developed over time to provide security even when the adversary learns certain secret values. In this work, we advance the modelling of security for AKE protocols by considering more granular partial leakage of long-term secrets of protocol participants: in our modified version of the extended CanettiKrawczyk (eCK) security model, the adversary can adaptively request arbitrary leakage of long-term secrets even after the test session is activated. Our modified eCK model can be instantiated as a bounded leakage model or continuous leakage model.

We present generic and concrete constructions of two-pass leakage-resilient key exchange protocols that are secure in the proposed security models. We introduce a new concept: the leakage-resilient NAXOS trick, which will be used to construct a leakage-resilient key exchange protocol. The leakage-resilient NAXOS trick computationally combines the long-term secret key and the ephemeral secret-key of a protocol participant, in such a way that even though the long-term secret key is partially leaked and the ephemeral secret key is fully leaked, the output of the NAXOS computation is indistinguishable from a random value to the adversary. We identify a special property for public-key cryptosystems: pair generation indistinguishability, and show how to obtain the leakage-resilient NAXOS trick

from a pair generation indistinguishable leakage-resilient public-key encryption scheme.

Compression is desirable for network applications as it saves bandwidth; however, when data is compressed before being encrypted, the amount of compression leaks information about the amount of redundancy in the plaintext. This side channel has led to the successful CRIME and BREACH attacks on web traffic protected by the Transport Layer Security (TLS) protocol. The general guidance in light of these attacks has been to disable compression, preserving confidentiality but sacrificing bandwidth. In this work, we examine two techniques–heuristic separation of secrets and fixed-dictionary compression, for enabling compression while protecting high-value secrets, such as cookies, from attack. We model the security offered by these techniques with new definitions of cookie indistinguishability and cookie recovery security, and report on the amount of compressibility that these techniques can achieve.

Contents

List of Figures

List of Tables

Declaration

The work contained in this thesis has not been previously submitted for a degree or diploma at any higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

Signed: **Date:** 16/11/2015

Previously Published Material

The following papers have been published or presented, and contain material based on the content of this thesis.

1. Janaka Alawatugoda, Colin Boyd and Douglas Stebila. Modelling After-the-fact Leakage for Key Exchange. In *Proceedings of the 9th ACM Symposium in Information, Computer and Communications Security (ASIACCS 2014), pages 207-216, ACM Press, 2014. June 04-06, 2014, Kyoto, Japan.* (Available at IACR Cryptology ePrint Archive, Report: 2014/131)

2. Janaka Alawatugoda, Colin Boyd and Douglas Stebila. Continuous After-the-fact Leakage-Resilient Key Exchange. In *Proceedings of the 19th Australasian Conference in Information Security and Privacy (ACISP 2014), pages 258-273, LNCS vol. 8544, Springer, 2014. July 07-09, 2014, Wollongong, Australia.* (Available at IACR Cryptology ePrint Archive, Report: 2014/264)

3. Janaka Alawatugoda, Colin Boyd and Douglas Stebila. Protecting Encrypted Cookies from Compression Side-Channel Attacks. In *Proceedings of the 19th International Conference in Financial Cryptography and Data Security (FC 2015), pages 86-106, LNCS vol. 8975, Springer, 2015. January 26-29, 2015, San Juan, Puerto Rico, The United States of America.* (Available at IACR Cryptology ePrint Archive, Report: 2014/724)

4. Janaka Alawatugoda, Colin Boyd and Douglas Stebila. Continuous After-the-fact Leakage-Resilient eCK-secure Key Exchange. In *Proceedings of the 15th IMA International Conference Cryptography and Coding (IMACC 2015), pages 277-294, LNCS vol. 9496, Springer, 2015. December 15-17, 2015, Oxford, The United Kingdom.* (Available at IACR Cryptology ePrint Archive, Report: 2015/355)

Acknowledgements

I am grateful to the many people who have helped me throughout my PhD journey. First, I thank my supervisors; Dr. Douglas Stebila of the Queensland University of Technology–Australia and Professor Colin Boyd of the Norwegian University of Science and Technology–Norway, for their guidance and support over the years. I am also grateful for the mentorship of Professor Tatsuaki Okamoto of NTT Secure Platform Laboratories–Japan.

I have had many helpful discussions with my colleagues in the Information Security Discipline of the Queensland University of Technology. My research and studies have been supported by QUTHDR and QUTPRA Scholarships from the Queensland University of Technology. I appreciate the funding support from the Information Security Discipline, the School of Electrical Engineering and Computer Science, and the Australian Research Council (ARC) Discovery Project DP130104304 for various conference travels, and I am grateful to Dr. Douglas Stebila and Professor Colin Fidge for approving funds for these travels. I was supported by the Nippon Telegraph and Telephone (NTT) Corporation–Japan for my internship at NTT Secure Platform Laboratories in Tokyo, and I am grateful to Professor Tatsuaki Okamoto for arranging the internship for me. Moreover, I received student stipend/complementary registration support from the National Institute of Information and Communications Technology (NICT)–Japan, the International Association for Cryptologic Research (IACR) and the International Financial Cryptography Association (IFCA) for conference/summer school travels.

I am grateful to my thesis panel members Dr. Douglas Stebila, Professor Colin Boyd, A/Professor Xavier Boyen and Dr. Harold Bartlett for their valuable comments on the thesis. I would like to acknowledge my mother, father, father-in-law, late mother-in-law and my wife, who have supported and encouraged me. Last but not least, I acknowledge all my friends in Sri Lanka, Australia, Japan and the USA who helped me during my PhD candidature and various travels.

To everyone, who shared moments in my life ...

Chapter 1

Introduction

Contents

As the Internet developed, more and more individuals and organizations connect their computers and private networks to the Internet. Since the Internet is a publicly available resource, the security of the information exchanged via the Internet is not guaranteed. Therefore, ensuring the security of the information becomes an important task. For many years, confidentiality, integrity and availability (known as the CIA triad) are known as the core principles of the information security [McC91].

Cryptography is engaged with communication systems to enforce the security of the information by establishing a *secure channel* for communication. A secure channel assures that no third party can see or modify the actual messages that are being transferred. Particularly, in the real world scenario the TLS/SSL protocol suite is used for this purpose. First, the TLS/SSL handshake protocol exchanges

a *secret key* (session key). Thereafter the TLS/SSL record protocol uses that session key to *encrypt* the messages. During the handshake protocol, both parties agree on an algorithm to encrypt data in the TLS/SSL record layer.

In the key exchange phase, long-term secrets of communicating parties can be leaked to the adversary, and in the message transmission phase, the secret keys (session keys) as well as the messages can be leaked to the adversary, due to the *side-channel attacks*.

Objective of the Thesis in Brief. As mentioned above, there are two main things to be done, when establishing and using a secure channel for communication:

1. a secret key (session key) should be exchanged between the intended parties,
2. encrypt the messages using the session key.

Our main objective is to construct cryptographic primitives for secure channel establishment and communication, which are resilient to the side-channel attacks.

1.1 Side-Channel Attacks and Countermeasures

The revolutionary idea of side-channel attacks was first introduced by Kocher [Koc96] by presenting timing attacks on Diffie-Hellman, RSA, DSS and other implementations. Although the cryptographic schemes are designed in such a way that they are hard to break computationally, leaking information from the implemented system may give sufficient power to the adversary to break the system by recovering the secret parameters such as secret keys. There are various kinds of side channels available such as timing information, electromagnetic radiation, acoustic signals, visual or light signals and leaking information about power consumption. Since it is very difficult to fully stop the information leakage from cryptosystems, side-channel attacks become a huge threat for the security of cryptosystems [HR07]. Moreover, it is very useful to study different types of side-channel attacks in the literature, to understand how they extract information and proceed to extract the secrets from the targeted cryptosystem. In the following, we concentrate on timing attacks, power analysis attacks, electromagnetic emission based attacks and compression-based leakage attacks, to understand the nature of side channel attacks.

Timing Attacks. Timing attacks enable the attacker to reveal secrets from the cryptosystem by measuring the amount of time taken for different computations. Cryptosystems take different amount of time for different operations. If the attacker can get the different timing information from the targeted cryptosystem, the attacker may be able to determine what is happening inside the system. As mentioned before, Kocher [Koc96] stated that by measuring the amount of time required for secret key operations, attackers might be able to find fixed Diffe-Hellman exponents, factor RSA keys and break other cryptosystems. He also stated that these kinds of attacks are computationally inexpensive and often need only a known ciphertext. Actual systems are potentially at risk whenever attackers can get accurate timing measurements, especially when the amount of time required to perform an operation is variable and dependent on a secret key or protected value. Bernstein demonstrated an attack on the implementation of Advanced Encryption Standard (AES) algorithm and showed that the standard AES algorithm is vulnerable for cache-timing attacks [Ber05]. Even though the brute-force search takes thousands of years to attack AES, Bernstein's attack could be completed within days as long as the adversary has the power to study the cache-timing information from the targeted system.

Power Analysis Attacks. Power analysis attacks are initiated by measuring the power consumption of a cryptographic device which is used to implement the cryptosystem. Messerges et al. [MDS02] applied this idea to attack an actual smart card, which uses Data Encryption Standard (DES). They examined the noise characteristics of the power signals and developed an approach to model the signal-to-noise ratio. They showed that the signal-to-noise ratio can be significantly improved using a multiple-bit attack. It is not difficult to attach a device which can measure the power consumption of an ATM machine and sending that information to the adversary. By analyzing the information an adversary can reveal the secret keys of smart cards used on the ATM.

Electromagnetic (EM)-Emission-based Attacks. EM-emission-based attacks are another possible type of side-channel attack. Electric circuits generate EM radiation as they operate. Attackers can use these radiation emission and analyze them to extract the secrets of the cryptosystem. Hutter et al. [HMF07] discussed the effectiveness of EM emission based attacks on electronic passports and contact-less payment systems which use passive 13.56 MHz radio frequency

identification devices. They stated that these devices can be successfully attacked with less than 1000 EM traces. Unlike power analysis attacks, EM emission based attacks can be performed remotely. Receivers can be used to catch emission radiation and more powerful receivers can get information from distant systems.

Cold Boot Attacks. A cold boot attack is a type of side-channel attack in which an attacker physically accesses a system and retrieves secret keys from a running operating system after a cold reboot (boot process in which the system starts from a powerless state) to restart the system. This attack relies on the data remanence property of DRAM and SRAM to retrieve memory contents that remain readable in the seconds to minutes after power has been removed.

Compression-based Leakage. In 2002, Kelsey [Kel02] showed how compression can act as a form of side-channel leakage. If plaintext data is compressed before being encrypted, the length of the ciphertext reveals information about the amount of compression, which in turn can reveal information about the plaintext. Kelsey notes that this side channel differs from other types of side channels in two key ways: "it reveals information about the plaintext, rather than key material", and "it is a property of the algorithm, not the implementation". Kelsey's most powerful attack is an *adaptive chosen input attack*: if an attacker is allowed to choose inputs x that are combined with a target secret s and the concatenation $x\|s$ is compressed and encrypted, observing the length of the outputs can eventually allow the attacker to extract the secret s.

1.1.1 Local versus Remote Side-Channel Attacks.

Side-channel attacks can be mounted locally (local attacks) or remotely (remote attacks) from the targeted system. Local attacks need some kind of physical access to the target system or proximity. For instance, side-channel attacks such as power analysis attacks, EM-emission based attacks, cold boot attacks need physical access to the targeted system or proximity to capture the leakage. Alternatively, remote attacks can be mounted from a long distance. For instance, the side-channel attacks such as timing attacks and compression-based attacks can be mounted from remotely, as the adversary can measure response time or ciphertext lengths from a distance system.

1.1.2 Countermeasures for Side-Channel Attacks.

Considering the countermeasures of side-channel attacks, mainly there are two approaches. One is a hardware based approach where the focus is to design hardware that minimizes the leakage of secret information. Bernstein has proposed ideas to design CPUs which provides protection against timing attacks on cache memory, as well as many ideas to mask leaking timing information by software based AES implementations [Ber05]. Besides this there are other software based countermeasures which are mostly focusing on masking the leaking information. In a previous work [AJR11], I have presented three methods for masking leaking timing information: injecting some randomness to the leaking cache-timing information, dedicating cache portions to fetch data from different memory portions and pre-fetching from the memory before the algorithm accesses the required memory portions and hence inject random timing information and change the cache access pattern respectively. A few possible countermeasures against compression-based leakage attacks would be disabling the compression, randomizing the compressed output length, masking the secrets or randomizing the secrets per request [GHP13]. Obviously, those types of countermeasures protect systems only against some specific attacks that are known at the moment. Those countermeasures are known as ad-hoc solutions.

Above we discussed a few known side-channel attacks and possible countermeasures against them. There may be many unknown side-channel attacks as well. Therefore, it is important to defend against both known and unknown side-channel attacks.

1.2 Leakage-Resilient Cryptography

As discussed above side-channels leak some amount of information about the secret parameters to the adversary. The basic idea of leakage-resilient cryptography is, even though some leaking information is visible to the adversary, the security of the cryptographic scheme remains. Trying to stop the leakage is nearly impossible because electronic devices have their physical limitations.

Even though a cryptographic scheme may be proven secure in a strong security model which does not address leakage attacks, it is not possible to say anything about the security of the cryptographic scheme in an environment where the adversary is capable of obtaining leakage information. In order to analyze the

leakage resiliency of cryptographic schemes we need to construct security models where the adversary is given capability of obtaining leakage information.

Different leakage models have been introduced to capture side-channel attacks. They provide leakage based information to the adversary under different constraints. In order to achieve leakage resilience in a particular leakage model, the scheme should be proven secure even when the adversary is capable of accessing leakage information in that particular leakage model.

Continuous Leakage Model. In the pioneering work of Micali and Reyzin [MR04], a general framework was introduced to model the leakage that occurs when computation takes place on secret parameters. This framework relies on the assumption that "only computation leaks information". Further, Micali and Reyzin mentioned that leakage only occurs from the secret memory portions which are actively involved in computations, and the amount of leakage per occurrence is less than the size of the corresponding secret memory portion, hence bounded by a leakage parameter λ. The adversary is allowed to obtain leakage from an arbitrarily large number of computations, hence the overall leakage amount is *unbounded* and it can be larger than the size of the secret key. This leakage model addresses side-channel attacks such as timing attacks, power analysis attacks and EM emission based attacks, which obtain leakage of secret values whenever computations take place on them.

This model is suitable to analyze *stateful* leakage-resilient cryptographic schemes [DP08, Pie09a], where at the end of each i^{th} execution round a new secret key state sk_{i+1} is computed using the current secret key state sk_i. sk_{i+1} is going to be used as the secret parameter of the next execution round $i+1$. Before the i^{th} round an attacker chooses (adaptively) a leakage function f_i and after the execution of the round, it receives $f_i(sk_i)$, under the constrain that $|f_i(sk_i)| \leq \lambda$.

Bounded Leakage Model. Inspired by "cold boot" attacks Akavia, Goldwasser and Vaikuntanathan constructed a general framework to model memory attacks [AGV09]. The adversary can adaptively choose an efficiently computable arbitrary leakage function, f_i and send it to the leakage oracle. The leakage oracle gives $f_i(sk)$ to the adversary where sk is the secret key. The only restriction comes here is that $\Sigma|f_i(sk)| \leq \lambda$, where λ is the leakage parameter, which is smaller than the size of sk.

This model is suitable to analyze *stateless* leakage-resilient cryptographic schemes which are not using new secret states for each round.

After-the-fact Leakage. Leakage which happens after the challenge is given to the adversary is considered as after-the-fact leakage. In security experiments for public-key cryptosystems, the challenge to the adversary is, given a ciphertext, distinguish the corresponding plaintext. In key exchange security models, the challenge to the adversary is to identify the real session key of a chosen session from a random session key [BR93, CK01, LLM07]. In leakage models for public-key cryptosystems, after-the-fact leakage is the leakage which happens after the challenge ciphertext is given whereas in leakage-resilient key exchange security models, after-the-fact leakage is the leakage which happens after the session key is established.

Earlier leakage models only consider the leakage which happens before the challenge is given (before-the-fact leakage). Hence the adversary is not allowed to obtain leakage after the challenge is given. Recent leakage models facilitate more granular leakage by allowing the adversary to issue leakage functions, and obtain leakage even after the challenge is given, either under the bounded or continuous leakage assumptions as explained above.

For leakage-resilient public-key encryption there are three properties which may be important differentiators for the different models. One is whether the model allows access to decryption of chosen ciphertexts before (CCA1) and after (CCA2) the challenge is known. The second is whether the leakage allowed to the adversary is *continuous* or *bounded*. The third is whether the leakage is allowed only before the challenge ciphertext is known or also after the fact.

In earlier models, such as that of Naor and Segev [NS09], it was expected that although the adversary is given access to the decryption oracle (CCA2), the adversary cannot be allowed to obtain leakage after the challenge ciphertext is given. This is because the adversary can encode the decryption algorithm and challenge ciphertext with the leakage function and by revealing a few bits of the decrypted value of the challenge ciphertext trivially win the challenge. Subsequently, Halevi and Lin [HL11] introduced after-the-fact leakage-resilient semantic security (CPLA2) on public-key cryptosystems, in the bounded leakage model. In their security experiment, the adversary is not allowed to access the decryption oracle. Dziembowski and Faust [DF11] defined an adaptively-chosen ciphertext after-the-fact leakage (CCLA2) in which the adversary is allowed to

access the decryption oracle adaptively and obtain leakage information even after the challenge ciphertext is given. Furthermore, they allow continuous leakage, so the total leakage amount is unbounded.

Modelling Compression-based Plaintext Leakage. The compression-based leakage is somewhat different from the information leakage we discussed in Section 1.1. First, in this scenario the leakage is from the plaintext that is being encrypted, rather than from the long-term secret keys. Second, this leakage is due to compression, which only leaks the length of the resulting output. Thus, previous leakage-resilient approaches are not suitable to model compression-based side-channel attacks. Let pt be the plaintext that is being encrypted, and f be an efficiently computable leakage function. If we consider the leakage as the output of $f(pt)$, then according to the scenario of compression-based leakage, the leakage function f belongs to a specific class, that only outputs length information. Since the leakage information of the compression-based leakage attacks is in the form of length information, it is reasonable to model that leakage using a leakage function f from a specific class that only output length information.

1.3 Research Objective and Thesis Outline

In this section we explain the objective of our research and the outline of this thesis in detail.

1.3.1 Objective

Due to the side-channel attacks, the long-term secret keys (in the key exchange phase), the session keys and the plaintext messages (in the message transmission phase) can be partially leaked to the adversary. Much research has been carried out in analyzing the partial leakage of secret keys (session keys) for symmetric-key encryption schemes and constructing leakage-resilient symmetric-key encryption schemes [DP08, SPY13, Pie09b], which addresses the issue of side-channel attacks in the message transmission phase. Therefore, that area is of research is covered to some extent. In this research, we address leakage of long-term secret keys in the key exchange phase, compression-based leakage attacks in the data transmission phase, and construct necessary leakage-resilient cryptographic primitives. Therefore, this research and the previous works on leakage-resilient symmetric-key encryption,

together can be used to build a stronger leakage-resilient secure channel suite for communication.

Leakage of long-term secret keys in the key exchange. With the development of side-channel attacks, a necessity arises to develop cryptosystems in a leakage-resilient manner. Being one of the important cryptographic primitives, key exchange protocols were considered to be constructed in a leakage-resilient manner. Even though most of the current key exchange security models like Bellare-Rogaway (BR) model [BR93], Canetti-Krawczyk (CK) model [CK01], extended Canetti-Krawczyk (eCK) model [LLM07] address different adversarial capabilities, they do not address the partial leakage of long-term secret parameters due to the side-channel attacks. Those do not suffice for analyzing the security of existing key exchange protocols in a leaky environment. Thus, there is a necessity to design new leakage security models for key exchange. Using those leakage security models it is possible to design and analyze leakage-resilient key exchange protocols.

Our first objective is to study the leakage resiliency of key exchange protocols. In order to address leakage resilience of key exchange protocols, it is necessary to construct key exchange security models which allow the adversary to obtain leakage information about secret parameters of protocol participants. We will introduce various leakage features such as continuous leakage, bounded leakage, after-the-fact leakage into key exchange security models. Our ultimate goal is to construct a security model with strong leakage features, with no additional restrictions than existing strong security models such as eCK model (bounded/continuous and after-the-fact leakage in eCK-style security). This way we can introduce stronger security models allowing more granular partial leakage of long-term secrets.

Leakage of the plaintext messages in the data transmission. We noticed that constructing a leakage-resilient secure channel for data transmission will not be completed with leakage-resilient key exchange and leakage-resilient symmetric-key encryption. We have to consider another gap, which is how to address compression-based leakage of plaintext data, even if we have a symmetric-key encryption scheme which is resilient to the secret key leakage. Relatively less academic research has been carried out in that area, and it is hard to find a reasonable leakage model which addresses compression-based leakage.

Therefore, our second objective is to present reasonable models to address

compression-based leakage and construct schemes which combine compression and encryption, in a way that they can be proven secure in our compression-based leakage models.

1.3.2 Outline

The organization and main contributions of this thesis are as follows.

Chapter 2: This chapter contains the technical background that will be useful to understand the technical details in the rest of the thesis. We focus on mathematical background, computational assumptions and cryptographic tools which will be used in this thesis.

Chapter 3: This chapter contains material for understanding key exchange protocols and security models for key exchange protocols. We contribute a new protocol construction and prove its security in an existing security model (the eCK model). This protocol will serve as the basis for a construction in Chapter 5. Further, we give a brief introduction about a preliminary leakage security model for key exchange, the Moriyama-Okamoto model [MO11], which is available in the literature.
Portions of this chapter have appeared in the following publication.

- Janaka Alawatugoda, Colin Boyd and Douglas Stebila. Continuous After-the-fact Leakage-Resilient eCK-secure Key Exchange. In *Proceedings of the 15th IMA International Conference Cryptography and Coding (IMACC 2015), pages 277-294, LNCS vol. 9496, Springer, 2015. December 15-17, 2015, Oxford, The United Kingdom.* (Available at IACR Cryptology ePrint Archive, Report: 2015/355)

Chapter 4: This chapter contributes a new leakage security model for key exchange, which addresses continuous and after-the-fact leakage of long-term secret keys, while enforcing additional restriction to the freshness condition, compared to the eCK or Moriyama-Okamoto model. Further, we construct a new generic protocol which is proven secure in the proposed model, using any suitable leakage-resilient public-key encryption scheme. Thus, the contributions in this chapter provide an initial solution for continuous and after-the-fact leakage of long-term secret keys of key exchange protocol participants.

Portions of this chapter have appeared in the following publication.

- Janaka Alawatugoda, Colin Boyd and Douglas Stebila. Continuous After-the-fact Leakage-Resilient Key Exchange. In *Proceedings of the 19th Australasian Conference in Information Security and Privacy (ACISP 2014), pages 258-273, LNCS vol. 8544, Springer, 2014. July 07-09, 2014, Wollongong, Australia.* (Available at IACR Cryptology ePrint Archive, Report: 2014/264)

Chapter 5: This chapter contributes two after-the-fact leakage security models for key exchange: one addresses bounded leakage and the other one addresses continuous leakage of long-term secret keys. This model contains eCK-style freshness condition, which also appears in eCK or Moriyama-Okamoto model. The eCK-style freshness condition is interesting because it allows the adversary to learn many combinations of the long-term and the ephemeral secrets from the target session. Thus, it releases the additional restrictions we enforced for the freshness condition of the security model in Chapter 4.

First, we present a new generic protocol construction, using any suitable leakage-resilient public-key encryption scheme and any suitable leakage-resilient signature scheme. The generic protocol construction is proven secure in either the bounded or the continuous leakage security model (at this point we use a slightly weakened version of the bounded/continuous leakage eCK-style model, due to limitations of existing leakage-resilient signature schemes). Then, we give a new concrete construction of a key exchange protocol which can be proven secure in the proposed continuous leakage eCK-like model (now we use the full continuous leakage eCK-like model without any restriction), using a leakage-resilient storage scheme and its refreshing protocol. Thus, the contributions in this chapter provide solutions for continuous/bounded and after-the-fact leakage of long-term secret keys of key exchange protocol participants, with more powerful eCK-style security. Portions of this chapter have appeared in the following publications.

- Janaka Alawatugoda, Colin Boyd and Douglas Stebila. Modelling After-the-fact Leakage for Key Exchange. In *Proceedings of the 9th ACM Symposium in Information, Computer and Communications Security (ASIACCS 2014), pages 207-216, ACM Press, 2014. June 04-06, 2014, Kyoto, Japan.* (Available at IACR Cryptology ePrint Archive, Report: 2014/131)

- Janaka Alawatugoda, Colin Boyd and Douglas Stebila. Continuous After-the-fact Leakage-Resilient eCK-secure Key Exchange. In *Proceedings of the 15th IMA International Conference Cryptography and Coding (IMACC 2015), pages 277-294, LNCS vol. 9496, Springer, 2015. December 15-17, 2015, Oxford, The United Kingdom.* (Available at IACR Cryptology ePrint Archive, Report: 2015/355)

Chapter 6: This chapter contributes security notions to analyze compression-based leakage on symmetric-key encryption schemes. Then, we examine two possible mitigation techniques; separating secrets from user inputs and fixed-dictionary compression, and prove their security in new security notions. Further, we report on the amount of compressibility that they can achieve in an example application scenario. Thus, the contributions in this chapter provide solutions and open up new research directions in the context of compression-based leakage attacks on symmetric-key encryption schemes.
The content of this chapter has appeared in the following publication.

- Janaka Alawatugoda, Colin Boyd and Douglas Stebila. Protecting Encrypted Cookies from Compression Side-Channel Attacks. In *Proceedings of the 19th International Conference in Financial Cryptography and Data Security (FC 2015), pages 86-106, LNCS vol. 8975, Springer, 2015. January 26-29, 2015, San Juan, Puerto Rico, The United States of America.* (Available at IACR Cryptology ePrint Archive, Report: 2014/724)

Chapter 7: This chapter summarizes the results obtained in this thesis and discusses future research directions.

Appendix A Appendix A gives the source code of the experiments done related to Chapter 6, to report the compressibility of separating secrets from user inputs and fixed-dictionary compression mitigation techniques.

Chapter 2

Background

Contents

In this chapter we describe several background concepts that help to understand the rest of the thesis. We describe the mathematical background, computational assumptions and cryptographic tools which are useful to understand the rest of the thesis.

2.1 Mathematical Background

In this thesis, we assume that all the parties (including the adversary) involved in a cryptographic protocol are probabilistic polynomial-time (PPT) algorithms. We now define negligible function which is useful in measuring the advantage of adversary against a cryptographic challenge or a computationally hard problem.

Definition 2.1.1 (Negligible)**.** A function $\mu : \mathbb{N} \to \mathbb{R}$ is negligible, if for every positive polynomial $p(\cdot)$ there exists an $N \in \mathbb{N}$ such that for all $k > N$, $\mu(k) < \frac{1}{p(k)}$.

Hence, the advantage of an adversary against a cryptographic challenge or a computationally hard problem is called negligible, if the advantage is negligible in the given security parameter k. A typical security parameter for a cryptographic algorithm is the length of its key.

2.1.1 Algebra

Definition 2.1.2 (Group)**.** A group $\mathbb{G}$ is a set along with an operation $\odot$ which satisfies the following axioms:

G 1. $\odot$ is associative.

G 2. There is an element $e \in \mathbb{G}$, called the identity, such that $e \odot a = a = a \odot e$ for all a $\in \mathbb{G}$.

G 3. Each element in $\mathbb{G}$ is invertible: for each $s \in \mathbb{G}$, there exists $b \in \mathbb{G}$ such that $a \odot b = b \odot a = e$.

Definition 2.1.3 (Abelian Group)**.** A group $\mathbb{G}$ is abelian if the operation $\odot$ is also commutative.

Definition 2.1.4 (Order of a Group)**.** The order of a group $\mathbb{G}$, denoted $|\mathbb{G}|$, is the number of elements in $\mathbb{G}$.

Definition 2.1.5 (Finite Group)**.** If the order is finite, then $\mathbb{G}$ is said to be a finite group.

Sometimes we write groups using multiplicative notion, where the operation $\odot$ is $\times$.

Definition 2.1.6 (Subgroup)**.** Let $\mathbb{G}$ is a group with operation $\odot$. Let $\mathbb{H}$ be a subset of $\mathbb{G}$. Then $\mathbb{H}$ be a subgroup of $\mathbb{G}$ if $\mathbb{H}$ is closed under $\odot$, $\mathbb{H}$ contains the identity of $\mathbb{G}$, and $\mathbb{H}$ is closed under inversion.

Definition 2.1.7 (Cyclic Subgroup)**.** $\mathbb{H}$ is a cyclic subgroup of $\mathbb{G}$ if there exists some element $x \in \mathbb{G}$ such that,

$$\mathbb{H} = \{\dots, x^{-2}, x^{-1}, 1, x, x^2, \dots\}$$

where we have used multiplicative notation for the group. We say that $\mathbb{H}$ is the subgroup of $\mathbb{G}$ generated by x and write $\mathbb{H} = \langle x \rangle$; x is called a generator of $\mathbb{H}$.

Definition 2.1.8 (Order of an Element)**.** For every $x \in \mathbb{G}$, $< x >$ is a subgroup of $\mathbb{G}$. The order of an element $x \in \mathbb{G}$, denoted $ord_{\mathbb{G}}(x)$, is the order of $\langle x \rangle$.

2.1.2 Number Theory

Definition 2.1.9 (Greatest Common Divisor)**.** For two integers a, b the greatest common divisor is denoted $\gcd(a, b)$, which is the largest integer g such that g divides a and g divides b. The two integers a and b are said to be coprime if $\gcd(a, b) = 1$.

Definition 2.1.10 (The Euler Function)**.** This is also called as Euler totient function, denoted by ϕ, gives the number of positive integers less than or equal to n that are coprime to n:

$$\varphi(n) = |\{a : \gcd(a, n) = 1 \leq a \leq n\}| \ .$$

If n is a prime number, then $\varphi(n) = n - 1$. If $n = ab$, where $a, b \in \mathbb{Z}$ are coprime, then $\varphi(n) = \varphi(a)\varphi(b)$.

Definition 2.1.11. A set of integers modulo a positive integer n is denoted $\mathbb{Z}/n\mathbb{Z}$ or $\mathbb{Z}_n$.

2.2 Computational Assumptions

We now describe some computational assumptions which form the basis of security for most cryptographic primitives.

Discrete Logarithm Assumption

Let k be the security parameter, $\mathcal{G}$ be a group generation algorithm and $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^k)$, where $\mathbb{G}$ is a cyclic group of prime order q and g is an arbitrary generator of $\mathbb{G}$. The discrete logarithm of $g^a \in \mathbb{G}$ to the base g is the unique integer $a \in \mathbb{Z}_q$. The discrete logarithm problem (DLP) is computing the discrete logarithm of g^a to the base g, given a random instance (g, g^a).

We say that discrete logarithm assumption holds in $\mathbb{G}$ if for all PPT algorithms, the probability of solving the DLP in $\mathbb{G}$ is negligible for a given security parameter k.

2.2.1 Diffie-Hellman Problems

Computational Diffie-Hellman (CDH) Assumption

Let k be the security parameter, $\mathcal{G}$ be a group generation algorithm and $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^k)$ as described above. The CDH problem is to compute g^{ab} given a random instance (g, g^a, g^b) for $a, b \in \mathbb{Z}_q$.

We say that computational Diffie-Hellman assumption holds in $\mathbb{G}$ if for all PPT algorithms $\mathcal{A}$, the probability of solving the CDH problem in $\mathbb{G}$ given as,

$$\Pr_{g,q}^{\mathrm{CDH}}(\mathcal{A}) = \Pr\left(\mathcal{A}(\mathbb{G}, g, q, g^a, g^b) = g^{ab}\right)$$

is negligible for a given security parameter k.

Decisional Diffie-Hellman (DDH) Assumption

Let k be the security parameter, $\mathcal{G}$ be a group generation algorithm and $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^k)$ as described above. Consider the following two distributions:

$$\mathcal{DH}_{\mathbb{G}} = \{(g, g^a, g^b, g^{ab}); a, b \xleftarrow{\$} \mathbb{Z}_q\}$$

and

$$\mathcal{R}_{\mathbb{G}} = \{(g, g^a, g^b, g^c); a, b, c \xleftarrow{\$} \mathbb{Z}_q\} \ .$$

It is said that DDH assumption holds in $\mathbb{G}$ if for all PPT algorithms $\mathcal{A}$, the advantage in distinguishing the two distributions $\mathcal{DH}$ and $\mathcal{R}$ given as,

$$\mathrm{Adv}^{\mathrm{DDH}}_{g,q}(\mathcal{A}) = \Big| \Pr[\mathcal{A}(\mathcal{DH}_{\mathbb{G}}) = 1] - \Pr[\mathcal{A}(\mathcal{R}_{\mathbb{G}}) = 1] \Big|$$

is negligible for a given security parameter k.

Gap Diffie-Hellman (GDH) Assumption

Let $\mathbb{G}, q, g$ be as described above. Given a random instance (g, g^a, g^b) for $a, b \in \mathbb{Z}_q$, the GDH problem is to find g^{ab} given an oracle $\mathcal{O}$ that solves the Decisional Diffie-Hellman problem in $\mathbb{G}$.

It is said that GDH assumption holds in $\mathbb{G}$ if for all PPT algorithms $\mathcal{A}$, the probability of solving the GDH problem is given as,

$$\Pr^{\mathrm{GDH}}_{g,q}(\mathcal{A}) = \Pr\left(\mathcal{A}(\mathbb{G}, g, q, \mathcal{O}, g^a, g^b) = g^{ab}\right)$$

is negligible for a given security parameter k.

Oracle Diffie-Hellman (ODH) Assumption [ABR01]

Let $\mathbb{G}, q, g$ be as described above, and H be arbitrary efficiently computable function. A PPT adversary $\mathcal{R}$ is interacting with the Oracle Diffie-Hellman challenger, which is defined using the following game:

- $u, v \xleftarrow{\$} \mathbb{Z}_q$
- $Z_0 \leftarrow \mathrm{H}(g^{uv}); Z_1 \xleftarrow{\$} \{0,1\}^k$
- $b \xleftarrow{\$} \{0,1\}$
- $b' \leftarrow \mathcal{R}^{\mathcal{O}^{\mathrm{ODH}}}(g^u, g^v, Z_b)$
- $\mathcal{R}$ wins if $b' = b$

$\underline{\mathcal{O}^{\mathrm{ODH}}\ \textbf{Oracle}}$

- If $X = g^u$, return $\perp$
- Else return $\mathrm{H}(X^v)$

It is said that ODH assumption holds in $\mathbb{G}$ if for all PPT algorithms $\mathcal{R}$, the the advantage of winning the ODH challenge is negligible in k.

In the security proof of protocol π in chapter 5, we will use this ODH challenger, where the function H is replaced by a key derivation function KDF.

2.3 Cryptographic Tools

We now present the cryptographic tools which will be useful in understanding the later chapters of this thesis.

2.3.1 The Random Oracle Model

In the random oracle model, it is assumed that there is an *oracle* which is capable of answering each distinct query with a truly random value, and any query which is previously asked from the random oracle will be answered with the exact same value. This is an ideal world assumption rather than a real world assumption. Actually, it is assumed that the random oracle can answer any query within polynomial time. Every party, even the adversary, has access to the random oracle.

Random oracles are widely used in cryptography as abstract functions. By using random oracles, proofs can often be done efficiently and cleanly. The main limitation of the random oracle model is that no function computable by a finite algorithm can implement a true random oracle. Nevertheless, for any natural protocol a proof of security in the random oracle model gives very strong evidence of the practical security of the protocol; If a protocol is proven secure in the random oracle model, attacks to the protocol must either be outside what was proven, or break one of the assumptions in the proof. In the real world implementations random oracles are often replaced with a hash function. Then, to break the random oracle assumption, one must discover some unknown and undesirable property of the actual hash function. For good hash functions where such properties are believed unlikely, therefore the considered protocol can be considered secure.

2.3.2 Public-Key Encryption

A public-key encryption scheme consists of three algorithms as follows:

- KG: This is a PPT algorithm that takes as input the security parameter and outputs a public/secret key pair (pk, sk). This also specifies the message (plaintext) space $\mathcal{M}$ and the ciphertext space $\mathcal{C}$.
- Enc: This is a PPT algorithm that takes as input $m \in \mathcal{M}$ and a public-key pk, and outputs a ciphertext $c \in \mathcal{C}$.

- Dec: This is a deterministic algorithm that takes as input a ciphertext $c \in \mathcal{C}$ and a secret key sk, and outputs either a message $m \in \mathcal{M}$ or the error symbol $\perp$.

A public-key encryption scheme must satisfy the correctness property: for all valid key pairs (pk, sk), if $c = \text{Enc}_{pk}(m)$ for any $m \in \mathbb{M}$, then $\text{Dec}_{sk}(c) = m$.

We now review two security notions for public-key encryption schemes: *indistinguishability against adaptive chosen ciphertext attacks* (IND-CCA2) and *indistinguishability against adaptive chosen plaintext attacks* (IND-CPA), referring Bellare et al. [BDPR98].

Definition 2.3.1 (Indistinguishability against Adaptive Chosen Ciphertext Attacks (IND-CCA2))**.** Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be any PPT adversary in the security parameter k, against a public-key encryption scheme PKE = (KG, Enc, Dec). The IND-CCA2 security experiment for the public-key encryption scheme PKE, $\text{Exp}^{\text{IND-CCA2}}_{\text{PKE},\mathcal{A}}(1^k)$, is defined as follows:

1. $(pk, sk) \xleftarrow{\$} \text{KG}(1^k)$
2. $(m_0, m_1, state) \leftarrow \mathcal{A}_1^{\text{Dec}(sk,c)}(pk)$ *such that* $|m_0| = |m_1|$
3. $b \xleftarrow{\$} \{0, 1\}$
4. $c^* \leftarrow \text{Enc}(pk, m_b)$
5. $b' \leftarrow \mathcal{A}_2^{\text{Dec}(sk,c)_{c^* \neq c}}(pk, c^*, state)$
6. $\mathcal{A}$ wins if $b' = b$

Decryption Oracle

- $\text{Dec}(sk, c) \rightarrow m$ where m is the corresponding plaintext c.
- returns m to $\mathcal{A}$

The public-key encryption scheme PKE is IND-CCA2-secure, if for every PPT adversary $\mathcal{A}$ the advantage of winning the security experiment $\text{Exp}^{\text{IND-CCA2}}_{\text{PKE}}(\mathcal{A})$: $\text{Adv}^{\text{IND-CCA2}}_{\text{PKE}}(\mathcal{A})$, is negligible in the security parameter k.

Definition 2.3.2 (Indistinguishability against Adaptive Chosen Plaintext Attacks (IND-CPA))**.** Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be any PPT adversary in the security parameter k, against a public-key encryption scheme PKE = (KG, Enc, Dec). The IND-CPA security experiment for the public-key encryption scheme PKE, $\text{Exp}^{\text{IND-CPA}}_{\text{PKE},\mathcal{A}}(1^k)$, is defined as follows:

1. $(pk, sk) \xleftarrow{\$} \text{KG}(1^k)$

2. $(m_0, m_1, st) \leftarrow \mathcal{A}_1(pk)$ *such that* $|m_0| = |m_1|$

3. $b \xleftarrow{\$} \{0,1\}$

4. $c^* \leftarrow \mathrm{Enc}(pk, m_b)$

5. $b' \leftarrow \mathcal{A}_2(pk, c^*, st)$

6. $\mathcal{A}$ wins if $b' = b$

The public-key encryption scheme PKE is IND-CPA-secure, if for every PPT adversary $\mathcal{A}$ the advantage of winning the security experiment $\mathrm{Exp}^{\text{IND-CPA}}_{\text{PKE},\mathcal{A}}(1^k)$: $\mathrm{Adv}^{\text{IND-CPA}}_{\text{PKE}}(\mathcal{A})$, is negligible in the security parameter k.

2.3.3 Digital Signature

A digital signature scheme SIG = (KG, Sign, Vfy) consists of three algorithms as follows:

- KG: This is a PPT algorithm that takes as input the security parameter 1^k and outputs a public-verification/secret-signing key pair (vk, sk). This also specifies the message (plaintext) space $\mathcal{M}$ and the signature space $\mathcal{S}$.
- Sign: This is a PPT algorithm that takes as input $m \in \mathcal{M}$ and a secret signing key sk, and outputs a signature $\sigma \in \mathcal{S}$.
- Vfy: This is a deterministic algorithm that takes as input a signature $\sigma \in \mathcal{S}$ and a public verification key vk, and outputs a boolean value: *true* if σ is a valid signature under vk or *false* otherwise.

We now describe the security notion called *existential unforgeability under chosen message attacks* (UFCMA).

Definition 2.3.3 (Existential Unforgeability under Chosen Message Attacks (UFCMA))**.** Let $\mathcal{A}$ be any PPT adversary in the security parameter k, against signature scheme SIG = (KG, Sign, Vfy).

A signature scheme SIG = (KG, Sign, Vfy) is UFCMA-secure if the advantage of any PPT adversary $\mathcal{A}$ in the following game: $\mathrm{Adv}^{\text{UFCMA}}_{\text{SIG}}(\mathcal{A})$, is negligible in the security parameter k.

1. $(sk, vk) \xleftarrow{\$} \mathrm{KG}(1^k)$

2. $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\text{Sign}_{m^* \neq m}(sk,m)}(vk)$

3. If $\text{Vfy}(vk, m^*, \sigma^*) = true$, then $\mathcal{A}$ wins

2.3.4 Key Derivation Functions

We review the definitions of key derivation functions by Krawczyk [Kra08]. Secure and efficient key derivation functions are available in the literature, for example based on HMAC [Kra08].

Definition 2.3.4 (Key Derivation Function)**.** Let k be the security parameter. A key derivation function KDF is an efficient algorithm that accepts as input four arguments: a value σ sampled from a source of keying material Σ, a length value ℓ and two additional arguments, a salt value r defined over a set of possible salt values and a context variable c, both of which are optional i.e., can be set to a null. The KDF output is a string of ℓ bits.

Definition 2.3.5 (Source of Key Material)**.** A source of keying material Σ is a two-valued (σ, κ) probability distribution generated by an efficient probabilistic algorithm, where σ is the secret source key material to be input to the KDF and κ is some public knowledge about σ or its distribution.

Definition 2.3.6 (Security of key derivation function with respect to a source of key material)**.** A key derivation function KDF is said to be secure with respect to a source of key material Σ, if no feasible attacker $\mathcal{B}$ can win the following distinguishing game with probability significantly better than 1/2. In other words the advantage: $\text{Adv}_{\text{KDF}}(\mathcal{B})$ of winning the following game is negligible in k.

1. $(\sigma, \kappa) \xleftarrow{\$} \Sigma(1^k)$. (Both the probability distribution as well as the generating algorithm have been referred to Σ)

2. A salt value r is chosen at random from the set of possible salt values defined by KDF (r may be set to a constant or a null value if so defined by KDF).

3. The attacker $\mathcal{B}$ is provided with κ and r.

4. $\mathcal{B}$ chooses arbitrary value ℓ and c.

5. A bit $b \xleftarrow{\$} \{0, 1\}$ is chosen at random. If $b = 0$, attacker $\mathcal{B}$ is provided with the output of $\text{KDF}(\sigma, r, \ell, c)$ else $\mathcal{B}$ is given a random string of ℓ bits.

6. $\mathcal{B}$ outputs a bit $b' \leftarrow \{0,1\}$. $\mathcal{B}$ wins if $b' = b$.

In our standard model protocol construction in Chapter 5, we will use the key derivation function of Krawczyk [Kra08] to derive the shared secrets in the protocol principals. There we will use σ as the secret value which the protocol principals exchanged (Diffie-Hellman value).

2.3.5 Pseudo Random Functions

Following we review the security definition of pseudo random function according to Katz and Lindell [KL07].

Definition 2.3.7 (Pseudo Random Functions)**.** Let $F : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$ be an efficient, length-preserving, keyed function. F is a *pseudo random function* if for all PPT distinguishers J, there is a negligible function *negl* such that:

$$\Big| \Pr[\mathcal{J}^{F(key,\cdot)}(1^k) = 1] - \Pr[\mathcal{J}^{f_{rnd}(\cdot)}(1^k) = 1] \leq negl(k) \Big|,$$

where the first probability is taken over uniform choice of $key \in \{0,1\}^k$ and the randomness of $\mathcal{J}$, and the second probability is taken over uniform choice of f_{rnd} and randomness of $\mathcal{J}$, and $\mathcal{J}$ is not given a key key.

We will use the pseudo random function to derive the session key, in our standard model protocol constructions in Chapter 4 and 5. In security proofs, we use an Oracle$^{\text{PRF}}$ which, given an input value, computes an output value either using F or f_{rnd}. Task of the distinguisher $\mathcal{J}$ is to distinguish whether the output is computed using F or f_{rnd}.

2.4 Leakage-Resilient Primitives

In this section we discuss the leakage-resilient cryptographic constructions, which we will use for the protocol constructions in this thesis.

2.4.1 Leakage-Resilient Storage

We review the definitions of leakage-resilient storage according to Dziembowski and Faust [DF11]. The idea behind their construction is to split the storage of elements into two parts using a randomized encoding function. As long as leakage

is limited from each of its two parts, no adversary can learn useful information about an encoded element. The key observation of Dziembowski and Faust is then to show how such encodings can be *refreshed* in a leakage-resilient way so that the new parts can be re-used. To construct a continuous leakage-resilient primitive the relevant secrets are split, used separately, and then refreshed between any two usages.

Definition 2.4.1 (Dziembowski-Faust leakage-resilient storage scheme)**.** For any $m, n \in \mathbb{N}$, the storage scheme $\Lambda^{n,m}_{\mathbb{Z}^*_q} = (\text{Encode}^{n,m}_{\mathbb{Z}^*_q}, \text{Decode}^{n,m}_{\mathbb{Z}^*_q})$ efficiently stores elements $s \in (\mathbb{Z}^*_q)^m$ where:

- $\text{Encode}^{n,m}_{\mathbb{Z}^*_q}(s) : s_L \xleftarrow{\$} (\mathbb{Z}^*_q)^n \backslash \{(0^n)\}$, then $s_R \leftarrow (\mathbb{Z}^*_q)^{n \times m}$ such that $s_L \cdot s_R = s$ and outputs (s_L, s_R).
- $\text{Decode}^{n,m}_{\mathbb{Z}^*_q}(s_L, s_R)$: outputs $s_L \cdot s_R$.

In the model we expect an adversary to see the results of a leakage function applied to s_L and s_R. This may happen each time computation occurs.

Definition 2.4.2 ($\boldsymbol{\lambda}$-limited adversary)**.** If the amount of leakage obtained by the adversary from each of s_L and s_R is limited to $\boldsymbol{\lambda} = (\lambda_1, \lambda_2)$ bits in total respectively, the adversary is known as a $\boldsymbol{\lambda}$-*limited adversary.*

Definition 2.4.3 ($(\boldsymbol{\lambda}_\Lambda, \epsilon_1)$-secure leakage-resilient storage scheme)**.** We say $\Lambda = (\text{Encode}, \text{Decode})$ is $(\boldsymbol{\lambda}_\Lambda, \epsilon_1)$-secure leakage-resilient, if for any $s_0, s_1 \xleftarrow{\$} \mathcal{M}$ and any $\boldsymbol{\lambda}_\Lambda$-limited adversary $\mathcal{C}$, the leakage from $\text{Encode}(s_0) = (s_{0L}, s_{0R})$ and $\text{Encode}(s_1) = (s_{1L}, s_{1R})$ are statistically ϵ_1-close. For an adversary-chosen leakage function $\mathbf{f} = (f_1, f_2)$, and a secret s such that $\text{Encode}(s) = (s_L, s_R)$, the leakage is denoted as $\big(f_1(s_L), f_2(s_R)\big)$.

Lemma 2.4.1 ([DF11])**.** Suppose that $m < n/20$. Then $\Lambda^{n,m}_{\mathbb{Z}^*_q} = (\text{Encode}^{n,m}_{\mathbb{Z}^*_q}, \text{Decode}^{n,m}_{\mathbb{Z}^*_q})$ is $(\boldsymbol{\lambda}, negl(n))$-secure for some negligible function $negl$ and $\boldsymbol{\lambda} = (0.3 \cdot n \log q, 0.3 \cdot n \log q)$.

The encoding function can be used to design different leakage resilient schemes with bounded leakage. The next step is to define how to *refresh* the encoding so that a continuous leakage is also possible to defend against.

Definition 2.4.4 (Refreshing of Leakage-Resilient Storage [DF11])**.** Let $(L', R') \leftarrow \text{Refresh}^{n,m}_{\mathbb{Z}^*_q}(L, R)$ be a refreshing protocol that works as follows:

- Input : (L, R) such that $L \in (\mathbb{Z}_q^*)^n$ and $R \in (\mathbb{Z}_q^*)^{n \times m}$.
- Refreshing R :
  1. $A \xleftarrow{\$} (\mathbb{Z}_q^*)^n \backslash \{(0^n)\}$ and $B \leftarrow$ non singular $(\mathbb{Z}_q^*)^{n \times m}$ such that $A \cdot B = (0^m)$.
  2. $M \leftarrow$ full rank $(\mathbb{Z}_q^*)^{n \times n}$ such that $L \cdot M = A$.
  3. $X \leftarrow M \cdot B$ and $R' \leftarrow R + X$.
- Refreshing L :
  1. $\tilde{A} \xleftarrow{\$} (\mathbb{Z}_q^*)^n \backslash \{(0^n)\}$ and $\tilde{B} \leftarrow$ full rank $(\mathbb{Z}_q^*)^{n \times m}$ such that $\tilde{A} \cdot \tilde{B} = (0^m)$.
  2. $\tilde{M} \leftarrow$ non-singular $(\mathbb{Z}_q^*)^{n \times n}$ such that $\tilde{M} \cdot R' = \tilde{B}$.
  3. $Y \leftarrow \tilde{A} \cdot \tilde{M}$ and $L' \leftarrow L + Y$.
- Output : (L', R')

Let $\Lambda = (\text{Encode}, \text{Decode})$ be a $(\boldsymbol{\lambda}_\Lambda, \epsilon_1)$-secure leakage-resilient storage scheme and Refresh be a refreshing protocol. We consider the following experiment Exp, which runs Refresh for ℓ rounds and lets the adversary obtain leakage in each round. For refreshing protocol Refresh, a $\boldsymbol{\lambda}_{\text{Refresh}}$-limited adversary $\mathcal{B}$, $\ell \in \mathbb{N}$ and $s \xleftarrow{\$} \mathcal{M}$, we denote the following experiment by $\text{Exp}_{(\text{Refresh},\Lambda)}(\mathcal{B}, s, \ell)$:

1. For a secret s, the initial encoding is generated as $(s_L^0, s_R^0) \leftarrow \text{Encode}(s)$.
2. For $j = 1$ to ℓ run $\mathcal{B}$ against the j^{th} round of the refreshing protocol.
3. Return whatever $\mathcal{B}$ outputs.

We require that the adversary $\mathcal{B}$ outputs a single bit $b \in \{0, 1\}$ upon performing the experiment Exp using $s \xleftarrow{\$} \{s_0, s_1\} \in \mathcal{M}$. Now we define leakage-resilient security of a refreshing protocol.

Definition 2.4.5 ($(\ell, \boldsymbol{\lambda}_{\text{Refresh}}, \epsilon_2)$-secure Leakage-Resilient Refreshing Protocol)**.** For a $(\boldsymbol{\lambda}_\Lambda, \epsilon_1)$-secure leakage-resilient storage scheme $\Lambda = (\text{Encode}, \text{Decode})$ with message space $\mathcal{M}$, Refresh is $(\ell, \boldsymbol{\lambda}_{\text{Refresh}}, \epsilon_2)$-secure leakage-resilient, if for every $\boldsymbol{\lambda}_{\text{Refresh}}$-limited adversary $\mathcal{B}$ and any two secrets $s_0, s_1 \in \mathcal{M}$, the statistical distance between $\text{Exp}_{(\text{Refresh},\Lambda)}(\mathcal{B}, s_0, \ell)$ and $\text{Exp}_{(\text{Refresh},\Lambda)}(\mathcal{B}, s_1, \ell)$ is bounded by ϵ_2.

Theorem 2.4.1 ([DF11]). Let $m/3 \leq n, n \geq 16$ and $\ell \in \mathbb{N}$. Let n, m and $\mathbb{Z}_q^*$ be such that $\Lambda_{\mathbb{Z}_q^*}^{n,m}$ is $(\boldsymbol{\lambda}, \epsilon)$-secure leakage-resilient storage scheme (Definition 2.4.1 and Definition 2.4.3). Then the refreshing protocol $\text{Refresh}_{\mathbb{Z}_q^*}^{n,m}$ (Definition 2.4.4) is a $(\ell, \boldsymbol{\lambda}/2, \epsilon')$-secure leakage-resilient refreshing protocol for $\Lambda_{\mathbb{Z}_q^*}^{n,m}$ (Definition 2.4.5) with $\epsilon' = 2\ell p(3p^m\epsilon + mp^{-n-1})$.

We will use the leakage-resilient storage scheme and its refreshing protocol in construction of the protocol in Table 5.2 in Chapter 5.

2.4.2 Adaptively Chosen Ciphertext After-the-fact Leakage Secure (CCLA2) Public-Key Cryptosystems

Dziembowski and Faust [DF11] constructed an adaptively chosen ciphertext after-the-fact leakage-resilient public-key cryptosystem, which is secure against continuous leakage. Following we review the CCLA2 security notion of Dziembowski and Faust, which we will use for the security proof of the protocol in Table 4.1 of Chapter 4.

Definition 2.4.6 (Security Against Adaptively Chosen Ciphertext After-the-fact Leakage Attacks (CCLA2)). This is a modification of the IND-CCA2 security notion. Let $k \in \mathbb{N}$ be the security parameter, λ be the leakage parameter and f_i be arbitrary, efficiently computable adaptive leakage functions. Let $\text{PKE} = (\text{KG}, \text{Enc}, \text{Dec})$ be a public-key encryption scheme, we define $\text{Adv}_{\text{PKE}}^{\text{CCLA2}}(\mathcal{D})$ as the advantage of any PPT adversary $\mathcal{D}$, winning the following game:

1. $(sk, pk) \xleftarrow{\$} \text{KG}(1^k)$.
2. $(m_0, m_1, state) \leftarrow \mathcal{D}^{\text{Dec}(sk,\cdot,f_i)}(pk)$ such that $|m_0| = |m_1|$
3. $b \xleftarrow{\$} \{0, 1\}$
4. $C \leftarrow \text{Enc}(pk, m_b)$
5. $b' \xleftarrow{\$} \mathcal{D}^{\text{Dec}_{\neq C}(sk,\cdot,f_i)}(C, state)$
6. Output b'. $\mathcal{D}$ wins if $b' = b$

Decryption Oracle

- $\text{Dec}(sk, c, f_i) \rightarrow (sk', m)$ where m is the corresponding plaintext of the ciphertext c and sk' is the update of the secret key sk.
- compute $f_i(sk)$ whenever $|f_i(sk)| \leq \lambda$
- Update the secret state sk to sk'
- returns $(m, f_i(sk))$ to $\mathcal{A}$

PKE is CCLA2-secure, if $\mathrm{Adv}^{\mathrm{CCLA2}}_{\mathrm{PKE}}(\mathcal{D})$ is negligible in k.

2.4.3 After-the-fact Leakage-resilient Semantically Secure (CPLA2) Public-Key Cryptosystems

Splitting the secret key into arbitrarily number of parts is known as the *split-state* model. The leakage is allowed from the splits of the secret key independently. The secret key is split into an arbitrary number $\tilde{n}$ parts such that $s = (s_1, \ldots, s_{\tilde{n}})$. The tuple leakage function $\mathbf{f} = (f_{1j}, \ldots, f_{\tilde{n}j})$ is an adversary chosen efficiently computable adaptive tuple leakage function, which consists of $\tilde{n}$ arbitrary leakage functions, and j indicates the j^{th} leakage occurrence. Each leakage function f_{ij} leaks $f_{ij}(s_i)$ from each s_i split of the secret key individually.

Halevi and Lin [HL11] constructed a generic after-the-fact leakage-resilient semantically secure public-key cryptosystem, which is secure against bounded leakage in the split-state model. It can be instantiated with the DDH-based leakage-resilient public-key cryptosystem of Naor and Segev [NS09]. Following we review the CPLA2 security notion of Halevi and Lin in the split-state model, which we will use for the security proof of the protocol in Table 5.1 of Chapter 5.

Definition 2.4.7 (After-the-fact Leakage-resilient Semantic Security (CPLA2))**.** This is a modification of the IND-CPA security notion. Let $k \in \mathbb{N}$ be the security parameter and $\boldsymbol{\lambda} = (\boldsymbol{\lambda}_{pre}, \boldsymbol{\lambda}_{post})$ be a tuple of two vectors, where $\boldsymbol{\lambda}_{pre} = (\lambda_{pre_1}, \ldots, \lambda_{pre_{\tilde{n}}})$ is the leakage bound vector before the challenge ciphertext is issued, and $\boldsymbol{\lambda}_{post} = (\lambda_{post_1}, \ldots, \lambda_{post_{\tilde{n}}})$ is the leakage bound vector after the challenge ciphertext is issued. Let $\mathbf{f}$ be the leakage function as described above. Let PKE = (KG, Enc, Dec) be a public-key cryptosystem, we define $\mathrm{Adv}^{\mathrm{CPLA2}}_{\mathrm{PKE}}(\mathcal{D})$ as the advantage of any probabilistic polynomial time (PPT) adversary $\mathcal{D}$, winning the following game:

1. $(s, p) \xleftarrow{\$} \mathrm{KeyGen}(1^k)$.
2. $(m_0, m_1) \leftarrow \mathcal{D}^{\mathrm{Leak}(\mathbf{f})}(p)$ such that $|m_0| = |m_1|$, for $i = 1, \ldots, \tilde{n}$, $\mathrm{Leak}(\mathbf{f})$ returns $f_{ij}(s_i)$ if $\sum_j |f_{ij}(s_i)| \leq \lambda_{pre_i}$.
3. $b \xleftarrow{\$} \{0, 1\}$.
4. $C \xleftarrow{\$} \mathrm{Enc}(pk, m_b)$.

5. $b' \leftarrow \mathcal{D}^{\text{Leak}(\mathbf{f})}(p, C)$ for $i = 1$ to $i = \tilde{n}$, Leak($\mathbf{f}$) returns $f_{ij}(s_i)$ if $\sum_j |f_{ij}(s_i)| \leq \lambda_{post_i}$.
6. $\mathcal{D}$ wins if $b' = b$.

PKE is CPLA2-secure, if $\text{Adv}_{\text{PKE}}^{\text{CPLA2}}(\mathcal{D})$ is negligible in k.

Note that when we use this scheme in Chapter 5, we will consider $\boldsymbol{\lambda} = \min(\boldsymbol{\lambda}_{pre}, \boldsymbol{\lambda}_{post})$ for simplicity.

2.4.4 Unforgeability Against Chosen Message Leakage Secure UFCMLA Signature Schemes

Katz et al. [KV09] constructed an unforgeability against chosen message leakage attacks secure signature scheme in bounded leakage model. It contains signing and verification operations based on NIZK proofs. Following we review the UFCMLA security notion of Katz et al., which we will use for the security proof of the protocol in Table 5.1 of Chapter 5.

Definition 2.4.8 (Unforgeability Against Chosen Message Leakage Attacks (UFCMLA))**.** This is a modification of the UFCMA security notion. Let $k \in \mathbb{N}$ be the security parameter, λ be the leakage parameter and f_i be arbitrary efficiently computable adaptive leakage functions. Let SIG = (KG, Sign, Vfy) be a signature scheme, we define $\text{Adv}_{\text{SIG}}^{\text{UFCMLA}}(\mathcal{E})$ as the advantage of any PPT adversary $\mathcal{E}$, winning the following game:

1. $r \xleftarrow{\$} \{0,1\}^*$
2. $(sk, vk) \xleftarrow{\$} \text{KG}(1^k, r); st \leftarrow r$
3. $(m^*, \sigma^*) \leftarrow \mathcal{E}^{\mathcal{O}^{\text{UFCMLA}}(\cdot,\cdot)}(vk)$
4. If $\text{Vfy}(vk, m^*, \sigma^*)$ = "true" and m^* is not been previously signed, then $\mathcal{E}$ wins.

$\mathcal{O}^{\text{UFCMLA}}(m, f_i)$

- $r_i \xleftarrow{\$} \{0,1\}^*$
- $\sigma \xleftarrow{\$} \text{Sign}(sk, m, r_i)$
- $st \leftarrow st \cup r_i$
- compute $\gamma \leftarrow f_i(st)$ whenever $\sum_i |f_i(st)| \leq \lambda$
- Return (σ, γ) to $\mathcal{E}$

SIG is UFCMLA-secure, if $\text{Adv}_{\text{SIG}}^{\text{UFCMLA}}(\mathcal{E})$ is negligible in k.

If we think about the split-state setting for the UFCMLA security notion, above notion is same as the case where $\tilde{n} = 1$, because the signing key sk has not been split.

Chapter 3

Key Exchange Protocols and Security Models

Contents

In 1976, Diffie and Hellman introduced a key exchange protocol [DH76], which enables two parties to exchange a secret key (session key) by communicating over a public channel.

Users *Alice* and *Bob* agree on a group $\mathbb{G}$ of prime order q and on a generator g of this group. This is done before executing the rest of the protocol, and g and q are assumed to be public. Alice picks a random integer $a \xleftarrow{\$} \mathbb{Z}_q$ and computes A as follows and sends it to Bob.

$$A = g^a$$

Bob picks a random integer $b \xleftarrow{\$} \mathbb{Z}_q$ and computes B as follows and sends it to Alice.

$$B = g^b$$

Then, Alice computes

$$B^a = (g^b)^a = s \in \mathbb{G}$$

and Bob computes

$$A^b = (g^a)^b = s \in \mathbb{G}$$

Finally, both Alice and Bob end up with the same value $s \in \mathbb{G}$. An eavesdropper who watches this communication can see A and B values, but should be unable to determine the values of s (assuming CDH holds).

Many key exchange protocols have been created based on the Diffie-Hellman key exchange primitive [BMP00,DvOW92,Jab96]. In these key exchange protocols, different types of keys may be used to compute session keys: long-term secret keys are the static secrets belong to the protocol participants which are often used to add authentication to the session key, ephemeral keys are the session specific secrets belong to protocol participants which are used to add freshness to the session key. There are number of known attacks against key exchange protocols:

Implicit Key Authentication. If a protocol provides a guarantee that no party apart from the protocol participants can compute the session key, that key exchange protocol is said to provide *implicit key authentication*. If a key exchange protocol provides implicit key authentication that protocol is said to be an *authenticated key exchange* protocol.

Key Confirmation. If a key exchange protocol provides a guarantee that each party is assured that all other participants possess the session key, that key exchange protocol is said to provide *key confirmation*.

Known Key Security. The knowledge of a session key should not enable the adversary to learn the session keys in other sessions; all session keys should not be depended on the session keys of other sessions.

Unknown Key Share (UKS) Security. It should not happen that a party A shares a session key with some party B, but believing that it is sharing the session key with some one else C. That means public keys and identities of the parties should be certified and confirmed or incorporated into protocol execution.

Key Compromise Impersonation (KCI) Resistance. Knowing the long-term secret key of a party A should not enable the adversary to impersonate other honest parties to A.

Forward Secrecy. An adversary who knows the long-term secret keys of parties should not be able to compute the session keys of past sessions between those two particular parties.

3.1 Key Exchange Security Models

In order to analyze the security of key exchange protocols, a formal methodology is needed. Therefore, key exchange security models have been created. A security model is a formal security statement of certain security features. Generally, security models are designed to reflect real world adversarial capabilities, addressing the known attacks (mentioned earlier). At the same time, it is natural to design security models with theoretical adversaries which have more capabilities than real world adversaries, because that way it is possible to address more powerful attacks which may exist in the future. Following is the general structure of a security model.

- **Definition of the algorithm:** Inputs, outputs and abstract description of the algorithm.
- **Adversary capabilities:** How the adversary can interact with the system and which information the adversary is allowed to learn, usually in the form of queries. As a usual practice the adversary is made as strong as possible by giving more capabilities to the adversary.

- **Security game:** The way in which the adversary perform queries.
- **Security goal:** The requirement for the adversary to win the security game.

In a security model, there is a predefined list of queries that an adversary can perform (adversary capabilities). Those queries reveal information such as session keys, ephemeral keys, long-term secret keys etc. Even after performing the queries, within the constraints defined in the security model, if the adversary's advantage of distinguishing the real session key from a random key chosen from the same distribution is negligible, the protocol is said to be secure in the particular security model. The session in which the adversary tries to distinguish the real session key from a random key, is known as the *target session.*

The Bellare-Rogaway models (BR93 [BR93], BR95 [BR95]), the Canetti-Krawczyk (CK) model [CK01], and the extended Canetti-Krawczyk (eCK) model [LLM07] are a few such security models, and protocol designers use them to analyze the security of key exchange protocols. We briefly look at a few of the earlier models such as the BR models and the CK model, and then discuss the eCK model in detail, as it is a more recent and widely used security model.

The Bellare-Rogaway Models. The BR93 model [BR93] is the first formal security model for key exchange protocols, in which the adversary is defined as a probabilistic polynomial time machine that initiates and controls all the communications between the protocol participants. Moreover, the adversary is allowed to *reveal* the session keys, learn the complete internal state of the protocol participants by *corrupting* the protocol participants, and overwrite the long-term secret key of the corrupted participant with any value of her choice. The Bellare-Rogaway models defined a notion for partner sessions. The definition of partner sessions is used in the security definition, to restrict the adversary's *reveal* and *corrupt* operations to sessions that are not partners of the target session. In the BR models security is defined on the notions of partner sessions and the indistinguishability of session keys.

The BR93 model defines the partnership using the notion of matching conversations, where a conversation is defined to be the sequence of messages sent and received by a particular protocol instance (session) of a protocol participant. In the BR95 model, the partnership is defined using the notion of a *partner function*, which uses the transcript containing the record of all messages sent and

received by a particular protocol instance (session) of a protocol participant, to determine the partner sessions. However, such partner definitions can easily go wrong [CBHM04].

Canetti-Krawczyk Model. In the CK model, the adversary is allowed to initiate and control all the communications between the protocol participants, reveal the session keys, learn the complete internal state of the protocol participants by corrupting the protocol participants and overwrite the long-term secret key of the corrupted participant with any value of her choice, as in the BR models, and additionally reveal the internal state of the protocol participants (but not the long-term secret keys). The partner sessions are defined using the notion of matching session identifiers (SIDs) and partner identifiers (PIDs). There is no formal definition of how SIDs should be defined and the values of SIDs are not specified in the CK model. It is assumed that the SIDs are known by the protocol participants before the protocol begins. Such an assumption may not be practical, because it requires some form of communication between the protocol participants before the protocol begins. The definition of partner sessions is used in the security definition, to restrict the adversary's session key reveal, session state reveal and corrupt operations to sessions that are not partners of the target session. In the CK model security is defined on the notions of partner sessions and the indistinguishability of session keys, which is similar to the security definition of the BR models.

In the BR models and the CK model, the adversary is not allowed to learn the long-term secret key of the owner of the target session, before it expires. Therefore, those models are not capable of addressing the key compromise impersonation attacks. Moreover, the BR models and the CK model not allow the adversary to reveal the session states or ephemeral keys of the target session or its partner session. Therefore, those models are not capable of addressing the ephemeral key leakage attacks. We will discuss the technical details of key exchange security models in detail, in section 3.1.1. We intentionally choose the eCK model for detailed discussion, because we use the eCK model as the foundation to build-up our leakage security models for key exchange protocols.

3.1.1 Extended Canetti-Krawczyk Model (eCK)

Parties and Long-term Keys

Let $\mathcal{U} = \{U_1, \ldots, U_{N_P}\}$ be a set of N_P parties. Each party U_i where $i \in [1, N_P]$ has a pair of long-term public and secret keys, (pk_{U_i}, sk_{U_i}). Each party U_i owns at most N_S number of protocol sessions.

Sessions

Each party may run multiple instances of the protocol concurrently or sequentially; we use the term *principal* to refer a party involved in a protocol instance, and the term *session* to identify a protocol instance at a principal. The notation $\Pi^s_{U,V}$ represents the s^{th} session at the owner principal U, with intended partner principal V. The principal which sends the first protocol message of a session is the *initiator* of the session, and the principal which responds to the first protocol message is the *responder* of the session. A session $\Pi^s_{U,V}$ enters an *accepted* state when it computes a session key. Note that a session may terminate without ever entering into the accepted state. The information of whether a session has terminated with or without acceptance is public.

Partnering

Legitimate execution of a key exchange protocol between two principals U and V makes two partnering sessions owned by U and V respectively. Two sessions $\Pi^s_{U,V}$ and $\Pi^{s'}_{U',V'}$ are said to be partners if all of the following hold:

1. both $\Pi^s_{U,V}$ and $\Pi^{s'}_{U',V'}$ have computed session keys;
2. messages sent from $\Pi^s_{U,V}$ and messages received by $\Pi^{s'}_{U',V'}$ are identical;
3. messages sent from $\Pi^{s'}_{U',V'}$ and messages received by $\Pi^s_{U,V}$ are identical;
4. $U' = V$ and $V' = U$;
5. Exactly one of U and V is the initiator and the other is the responder.

The protocol is said to be *correct* if two partner sessions compute identical session keys.

Adversarial Powers

The adversary $\mathcal{A}$ is a probabilistic polynomial time algorithm in the security parameter k, that has the control over the whole network. $\mathcal{A}$ interacts with set of sessions which represent protocol instances. $\mathcal{A}$ can adaptively ask following queries.

- `Send` (U, V, s, m) query- This query allows $\mathcal{A}$ to run the protocol. It sends the message m to the session $\prod_{U,V}^{s}$ as coming from the session $\prod_{V,U}^{s'}$. $\prod_{U,V}^{s}$ will return to $\mathcal{A}$ the next message according to the protocol conversation so far or decision on whether to accept or reject the session. $\mathcal{A}$ can also use this query to initiate a new protocol instance with blank m. This query captures capabilities of active adversary, who can initiate sessions and modify or delay protocol messages.

- `Session-Key reveal` (U, V, s) query- If a session $\prod_{U,V}^{s}$ has accepted and holds a session key, $\mathcal{A}$ gets the session key of $\prod_{U,V}^{s}$. A session can only accept a session key once. This query captures the known key attacks.

- `Ephemeral-Key reveal` (U, V, s) query- Gives all the ephemeral keys (per session randomness) of the session $\prod_{U,V}^{s}$ to $\mathcal{A}$.

- `Corrupt` (U) query- $\mathcal{A}$ gets all the long-term secrets of the principal U. But this query does not reveal any session keys to $\mathcal{A}$. This query captures the key compromise impersonation (KCI) attacks, unknown key share (UKS) attacks and forward secrecy.

- `Test` (U, s) query- Once a session $\prod_{U,V}^{s}$ has accepted and holds a session key, $\mathcal{A}$ can attempt to distinguish it from a random key. When $\mathcal{A}$ asks the `Test` query, the session $\prod_{U,V}^{s}$ first chooses a random bit $b \in \{0, 1\}$ and if $b = 1$, the actual session key is returned to $\mathcal{A}$, otherwise a random session key is chosen uniformly at random from the same session key distribution, and is returned to $\mathcal{A}$. This query is only allowed to be asked once.

Freshness

A session $\prod_{U,V}^{s}$ is said to be *fresh* if and only if all of the following hold:

1. The session $\prod_{U,V}^{s}$ and its partner (if it exists), $\prod_{V,U}^{s'}$ have not been asked the `Session- Key reveal` query.

2. If partner $\prod_{V,U}^{s'}$ exists none of the following combinations have been asked:
   (a) `Corrupt`(U) and `Ephemeral-Key reveal`(U, V, s)
   (b) `Corrupt`(V) and `Ephemeral-Key reveal`(V, U, s')
3. If partner $\prod_{V,U}^{s'}$ does not exist none of the following combinations have been asked
   (a) `Corrupt`(V)
   (b) `Corrupt`(U) and `Ephemeral-Key reveal`(U, V, s)

Security Game

- **Stage 0:** The challenger generates the keys by using the security parameter k.
- **Stage 1:** $\mathcal{A}$ is executed and may ask any of `Send`, `Session-Key reveal`, `Ephemeral-Key reveal` and `Corrupt` queries to any session at will.
- **Stage 2:** At some point $\mathcal{A}$ chooses a fresh session and asks the `Test` query.
- **Stage 3:** $\mathcal{A}$ may continue asking `Send`, `Session-Key reveal`, `Ephemeral-Key reveal` and `Corrupt` queries. The only condition is that $\mathcal{A}$ cannot violate the freshness of the test session.
- **Stage 4:** At some point $\mathcal{A}$ outputs the bit $b' \in \{0, 1\}$ which is its guess of the value b on the test session. $\mathcal{A}$ wins if $b' = b$.

Definition of Security

Let $\text{Succ}_{\mathcal{A}}$ be the event that the adversary $\mathcal{A}$ wins the eCK game.

Definition 3.1.1. A protocol (π) is said to be secure in the eCK model if there is no PPT adversary $\mathcal{A}$ who can win the eCK game with non-negligible advantage in the security parameter k. The advantage of an adversary $\mathcal{A}$ is defined as

$$\text{Adv}_{\pi}^{\text{eCK}}(\mathcal{A}) = |2\text{Pr}(\text{Succ}_{\mathcal{A}}) - 1| \ .$$

The essential difference between the eCK [LLM07] and the CK model is that the eCK model substitutes the adversarial operation of revealing the complete internal state of the protocol participants with a new operation to reveal the ephemeral secret key, which reveals the randomness used in the specified session. The important point to note is that the ephemeral key does not include the session state that has been computed using the long-term secret of the protocol participant. This is not the case in the CK model, in which the adversary is allowed access to all the inputs (including the randomness, but excluding the long-term secret itself) and the results of all the computations done as part of a session. In the CK model, the adversary is not allowed to reveal the session state of the target session or its partner session, whereas in the eCK model, the adversary is allowed to reveal both of the ephemeral keys of the target session, as long as the owner and the partner principals to the target session are not corrupted. Thus, the eCK model can address the ephemeral key reveal attacks.

In the CK model, after the target session has expired, the adversary is allowed to learn the long-term secret keys of the protocol participants of the target session, regardless of whether the adversary actively interfered with the target session. The eCK model only allows the adversary to learn the long-term secret keys of both protocol participants of the target session when the adversary has not actively interfered with the target session. Therefore, the CK model addresses the perfect forward secrecy, while the eCK model only addresses the weak perfect forward secrecy.

3.1.2 Leakage Security Models for Key Exchange Protocols: Moriyama-Okamoto Model

Moriyama and Okamoto have presented a suitable security model to capture leakage, and a proven secure protocol in that model [MO11]. The security model introduced by Moriyama and Okamoto is based on the eCK model. The Moriyama-Okamoto model allows the adversary to obtain leakage of a long-term secret key sk, of a protocol principal U, by issuing adaptively-chosen arbitrary leakage functions f_i and specifying the identity of the protocol principal U. Hence, in addition to the adversarial powers in the eCK, model Moriyama-Okamoto model provides:

- `StaticKeyLeakage`(f, U) query: From this query the adversary obtains

$f_i(sk)$ where sk denotes the long-term secret key of the principal U.

Further, it is important to study the constraints in the Moriyama-Okamoto model. We can identify two main limitations in Moriyama-Okamoto model.

1. The adversary is allowed to obtain leakage from long-term secrets of protocol participants; this leakage is bounded by some parameter λ.
2. The model does not allow the adversary to obtain leakage after the test session is activated.

Since the Moriyama-Okamoto model is restricted to the bounded leakage, it can only address the side-channel attacks such as cold boot attacks (to some extent), which happen due to the leakage of bounded amount of information from the secret memory. The Moriyama-Okamoto model allows the adversary to reveal either the long-term secret key or the ephemeral secret key of the target session (same as in the eCK model). Additionally, the the Moriyama-Okamoto model allows bounded amount of leakage of the long-term secret key with the ephemeral secret key reveal from the target session. Thus the Moriyama-Okamoto model addresses the cold boot attacks to some extent by addressing following situations: the attacker reveals either (i) the long-term secret key, (ii) the ephemeral secret key, or (iii) the ephemeral secret key and part of the long-term secret key of the target session.

Differently, side-channel attacks which happen due to the continuous leakage of secret keys, such as timing attacks or power analysis attacks can not be modelled using the Moriyama-Okamoto model. Because the Moriyama-Okamoto model does not address continuous leakage of the long-term secret keys, which happens whenever computations use the long-term secret keys. Further, restricting the leakage to occur only before the target session is activated is not a natural restriction. Therefore, although the Moriyama-Okamoto model addresses side-channel attacks for some extent, there is some gap between the Moriyama-Okamoto model and real world side-channel attacks.

Those two limitations are considered when defining the freshness of a session in the Moriyam-Okamoto model.

Moriyama-Okamoto Freshness

A session $\prod^s_{U,V}$ is λ-leakage fresh if the following conditions hold:

- $\prod_{U,V}^{s}$ is a fresh session in the sense of the eCK model.
- Before the adversary activates the session $\prod_{U,V}^{s}$, the total amount of leakage that the adversary obtains from each partner principal of the session $\prod_{U,V}^{s}$: U and V, is bounded by the leakage parameter λ.
- After the session $\prod_{U,V}^{s}$ is activated, no leakage is allowed from the partner principals of the session $\prod_{U,V}^{s}$.

Apart from the freshness condition, partnering and the security game are the same as in the eCK model.

3.2 Key Exchange Protocols

In this section we look at a few well-known key exchange protocols. All examples we consider here are two-party, two message-pass key exchange protocols, and all of them are based on the Diffie-Hellman key exchange protocol.

3.2.1 MQV Protocol

The MQV protocol [LMQ+98], shown in Table 3.1, is a key exchange protocol based on the Diffie-Hellman scheme and it is considered as a highly efficient protocol. Following we describe the message flow of MQV protocol. In this protocol, w_A and r_A are the long-term secret key and the ephemeral secret key (respectively) of party A, which W_A and R_A are the corresponding public keys of A. f denotes the bit length of n, the prime order of the base point P of a finite elliptic curve, i.e., $f = \lfloor \log_2 n \rfloor + 1$. If Q is a finite elliptic curve point, then $\bar{Q}$ is defined as follows: Let x_Q be the x-coordinate of Q, and let $\bar{x}_Q$ be the integer obtained from the binary representation of x_Q. Then $\bar{Q}$ is defined to be the integer $(\bar{x}_Q \mod 2^{\lceil f/2 \rceil} + 2^{\lceil f/2 \rceil})$.

The MQV protocol works on elliptic curve group. It is also possible to modify this to be working in an arbitrary finite group. The MQV protocol has not been formally proved in any security model.

3.2.2 HMQV Protocol

Krawczyk [Kra05] analyzed the MQV protocol and stated that the MQV fails to satisfy some of the most important security properties such as resistance

A		B
$w_A \xleftarrow{\$} \{0,1\}^n$		$w_B \xleftarrow{\$} \{0,1\}^n$
$W_A \leftarrow w_A P$		$W_B \leftarrow w_B P$
$r_A \xleftarrow{\$} \{0,1\}^n$		$r_B, \xleftarrow{\$} \{0,1\}^n$
$R_A \leftarrow r_A P$	$\xrightarrow{R_A}$	$R_B \leftarrow r_B P$
	$\xleftarrow{R_B}$	
$\boxed{s_A \leftarrow (r_A + \bar{R}_A w_A) \pmod n}$		$\boxed{s_B \leftarrow (r_B + \bar{R}_B w_B) \pmod n}$
$K \leftarrow h_{s_A}(R_B + \bar{R}_B W_B)$		$K \leftarrow h_{s_B}(R_A + \bar{R}_A W_A)$
K is the session key		

Table 3.1: MQV protocol

to known-key attacks, resistance to basic impersonation attacks, resistance to unknown key share (UKS) attacks, resistance to key compromise impersonation (KCI) attacks and fails to provide perfect forward security. Then he created another protocol which is known as "hashed-MQV" (HMQV). He claimed that the HMQV is a hashed variant of the MQV with improved security while preserving the performance of the MQV unchanged. The design and proof of HMQV is on challenge-response signature schemes, which have the property that both signer and challenger can compute the same signature.

The difference of the HMQV over the MQV comes in the boxed line of the MQV protocol shown in Table 3.1. In the MQV, the parties compute $\bar{R}_A = (\bar{x}_{R_A} \mod 2^{\lceil f/2 \rceil} + 2^{\lceil f/2 \rceil})$ and $\bar{R}_B = (\bar{x}_{R_B} \mod 2^{\lceil f/2 \rceil} + 2^{\lceil f/2 \rceil})$ whereas in the HMQV they compute $\bar{R}_A = \mathrm{H}(R_A, B)$ and $\bar{R}_B = \mathrm{H}(R_B, A)$ where H is a hash function modeled as a random oracle and A,B are identities of the intended partners. The HMQV protocol has been formally proven in a variant of the CK model named as the CK-HMQV model. We note that the proof for HMQV protocol is in random oracle model.

3.2.3 NAXOS Protocol

The NAXOS protocol [LLM07] was originally published with the eCK model, and it is proven secure in the eCK model. Table 3.2 shows the NAXOS protocol. Let $\mathbb{G}$ be a group of prime order with generator g. Here a and $\bar{A}$ are the long-term secret and public keys of A, while x and X are the ephemeral secret and public keys of A. The important feature we can see is that the long-term secret key and the ephemeral secret key are combined using a hash function H_1. The trick of combining the long-term secret key and the ephemeral secret key is known as

"NAXOS trick". In this protocol both hash functions H_1 and H are modeled as random oracles.

A		B
$a \xleftarrow{\$} \mathbb{Z}_q$		$b \xleftarrow{\$} \mathbb{Z}_q$
$\bar{A} \leftarrow g^a$		$\bar{B} \leftarrow g^b$
$x \xleftarrow{\$} \mathbb{Z}_q$		$y \xleftarrow{\$} \mathbb{Z}_q$
$\bar{x} \leftarrow \mathrm{H}_1(x, a)$		$\bar{y} \leftarrow \mathrm{H}_1(y, b)$
$X = g^{\bar{x}}$	$\xrightarrow{X}$	$Y = g^{\bar{y}}$
	$\xleftarrow{Y}$	
$\kappa = \mathrm{H}(Y^a, \bar{B}^{\mathrm{H}_1(x,a)}, Y^{\mathrm{H}_1(x,a)}, A, B)$		$\kappa = \mathrm{H}(\bar{A}^{\mathrm{H}_1(y,b)}, X^b, X^{\mathrm{H}_1(y,b)}, A, B)$
κ is the session key		

Table 3.2: NAXOS protocol

3.2.4 CMQV Protocol

Using the MQV, the HMQV and the NAXOS protocols Fujioka et al. [FSU09] created a new key exchange protocol called "Combined MQV" (CMQV). Table 3.3 shows the CMQV protocol. Let $\mathbb{G}$ be a group of prime order with generator g. Here a and $\bar{A}$ are the long-term secret and public keys of A, while x and X are the ephemeral secret and public keys of A. The main advantages of the CMQV protocol are, better efficiency than the NAXOS protocol and proven-security in the eCK model. The CMQV protocol uses the NAXOS trick to achieve the eCK security. As for the NAXOS protocol, the CMQV security proof also uses the random oracle assumption, because hash functions H_1, H_2 and H are modeled as random oracles.

3.3 eCK-Secure Key Exchange without NAXOS Trick: Protocol P1

The motivation of LaMacchia et al. [LLM07] in designing the eCK model was that an adversary should have to compromise both the long-term and ephemeral secret keys of a party in order to recover the session key.

Recently, some researchers worked on constructing eCK-secure key exchange protocols without NAXOS trick [MO09, Yan13]. The motivation for such research can be explained as follows: The exponent of the ephemeral public key can be

A		B
$a \xleftarrow{\$} \mathbb{Z}_q$		$b \xleftarrow{\$} \mathbb{Z}_q$
$\bar{A} \leftarrow g^a$		$\bar{B} \leftarrow g^b$
$x \xleftarrow{\$} \mathbb{Z}_q$		$y \xleftarrow{\$} \mathbb{Z}_q$
$\bar{x} = \mathrm{H}_1(x, a)$		$\bar{y} = \mathrm{H}_1(y, b)$
$X = g^{\bar{x}}$		$Y = g^{\bar{y}}$
	$\xrightarrow{(B,A,X)}$	
	$\xleftarrow{(A,B,X,Y)}$	
$E = \mathrm{H}_2(Y, A, B)$		$D = \mathrm{H}_2(X, A, B)$
$\sigma = (Y\bar{B}^E)^{\bar{x}+Da}$		$\sigma = (X\bar{A}^D)^{\bar{y}+Eb}$
$\kappa = \mathrm{H}(\sigma, X, Y, A, B)$		$\kappa = \mathrm{H}(\sigma, X, Y, A, B)$
κ is the session key		

Table 3.3: Two-pass CMQV protocol

leaked to the adversary at some point. But when we consider the eCK model, it does not address the leakage of the exponent of the ephemeral public key in a NAXOS-type protocol. Thus, it seems that there is an unnatural and indirect assumption of a leakage-free exponentiation computation or leakage-free random source, in the eCK-security proof of the NAXOS-type key exchange protocols. Therefore, eliminating the NAXOS trick and still preserving the eCK security would be more realistic.

3.3.1 Construction of Protocol P1

Influenced by the above motivation, we aim to construct a new eCK-secure key exchange protocol which does not use the NAXOS trick, but combines long-term secret keys and ephemeral secret keys to compute the session key, in a way that guarantees eCK security of the protocol. Thus, we construct the key exchange protocol P1 shown in Table 3.4, which is a Diffie-Hellman-type key agreement protocol. Let k be the security parameter and group $\mathbb{G}$ be generated using a group generation algorithm which takes k as an input, where $\mathbb{G}$ be a group of prime order q with generator g. Here a and A are the long-term secret and public keys of Alice, while x and X are the ephemeral secret and public keys of Alice. After exchanging the public values both principals compute a Diffie-Hellman-type shared secret, and then compute the session key using a random oracle H. We use the random oracle because otherwise it is not possible to perfectly simulate the interaction between the adversary and the protocol, in a situation where the simulator does not know a long-term secret key of a protocol principal. Since the

P1 protocol does not have a NAXOS computation, it does not contain a exponent value, for which the leakage is not addressed in the eCK model. We will use the P1 protocol as a building block for a leakage-resilient key exchange protocol in chapter 5.

Alice (Initiator)		Bob (Responder)
$a \xleftarrow{\$} \mathbb{Z}_q^*, A \leftarrow g^a$		$b \xleftarrow{\$} \mathbb{Z}_q^*, B \leftarrow g^b$
$x \xleftarrow{\$} \mathbb{Z}_q^*, X \leftarrow g^x$	$\xrightarrow{Alice,X}$	$y \xleftarrow{\$} \mathbb{Z}_q^*, Y \leftarrow g^y$
	$\xleftarrow{Bob,Y}$	
$Z_1 \leftarrow B^a, Z_2 \leftarrow B^x$		$Z_1' \leftarrow A^b, Z_2' \leftarrow X^b$
$Z_3 \leftarrow Y^a, Z_4 \leftarrow Y^x$		$Z_3' \leftarrow A^y, Z_4' \leftarrow X^y$
$K \leftarrow \mathrm{H}(Z_1, Z_2, Z_3, Z_4, Alice, X, Bob, Y)$		$K \leftarrow \mathrm{H}(Z_1', Z_2', Z_3', Z_4', Alice, X, Bob, Y)$
K is the session key		

Table 3.4: Concrete construction of Protocol P1

In order to compute the session key, the protocol P1 combines four components ($Z_1 \leftarrow B^a$, $Z_3 \leftarrow Y^a$, $Z_4 \leftarrow Y^x$, $Z_2 \leftarrow B^x$) using the random oracle function H. These four components cannot be recovered by the attacker without both the ephemeral and long-term secret keys of at least one protocol principal, which allows a proof of the eCK security.

3.3.2 Security Analysis of the Protocol P1

Theorem 3.3.1. If H is modeled as a random oracle and $\mathbb{G}$ is a group of prime order q and generator g, where the gap Diffie-Hellman (GDH) assumption holds, then the protocol P1 is secure in the eCK model.

Let $\mathcal{U} = \{U_1, \ldots, U_{N_P}\}$ be a set of N_P parties. Each party U_i owns at most N_S number of protocol sessions. Let $\mathcal{A}$ be any PPT adversary in the security parameter k, against the protocol P1. Then, $\mathcal{B}$ is an efficient algorithm which can be constructed using the adversary $\mathcal{A}$, against the GDH problem such that the advantage of $\mathcal{A}$ against the eCK-security of the protocol P1, $\mathrm{Adv}_{\mathrm{P1}}^{\mathrm{eCK}}$ is:

$$\mathrm{Adv}_{\mathrm{P1}}^{\mathrm{eCK}}(\mathcal{A}) \leq \max\Big(N_P^2 N_S^2\big(\mathrm{Pr}_{g,q}^{\mathrm{GDH}}(\mathcal{B})\big), N_P^2\big(\mathrm{Pr}_{g,q}^{\mathrm{GDH}}(\mathcal{B})\big), N_P^2 N_S\big(\mathrm{Pr}_{g,q}^{\mathrm{GDH}}(\mathcal{B})\big), N_P^2 N_S\big(\mathrm{Pr}_{g,q}^{\mathrm{GDH}}(\mathcal{B})\big), N_P^2 N_S\big(\mathrm{Pr}_{g,q}^{\mathrm{GDH}}(\mathcal{B})\big), N_P^2\big(\mathrm{Pr}_{g,q}^{\mathrm{GDH}}(\mathcal{B})\big)\Big) .$$

(in 6 different cases in the proof $\mathcal{B}$ denotes different simulations)

Proof. Let $\mathbf{A}$ denote the event that $\mathcal{A}$ wins the eCK challenge. Let $\mathbf{H}$ denote the event that $\mathcal{A}$ queries the random oracle H with $(\mathrm{CDH}(A^*, B^*), \mathrm{CDH}(B^*, X^*), \mathrm{CDH}(A^*, Y^*), \mathrm{CDH}(X^*, Y^*), \textit{initiator}, X, \textit{responder}, Y)$, where A^*, B^* are the long-term public-keys of the two partners to the test session, and X^*, Y^* are their ephemeral public keys for this session. Note that when $A = g^a, B = g^b, \mathrm{CDH}(A, B) = g^{ab}$; also *initiator* is the initiator of the session and *responder* is the responder of the session.

$$\Pr(\mathbf{A}) \leq \Pr(\mathbf{A} \wedge \mathbf{H}) + \Pr(\mathbf{A} \wedge \bar{\mathbf{H}}) \ .$$

Without the event $\mathbf{H}$ occurring, the session key given as the answer to the `Test` query is random-looking to the adversary, and therefore $\Pr(\mathbf{A}|\bar{\mathbf{H}}) = \frac{1}{2}$. $\Pr(\mathbf{A} \wedge \bar{\mathbf{H}}) = \Pr(\mathbf{A}|\bar{\mathbf{H}}) \Pr(\bar{\mathbf{H}})$, and therefore $\Pr(\mathbf{A} \wedge \bar{\mathbf{H}}) \leq \frac{1}{2}$. Hence,

$$\Pr(\mathbf{A}) \leq \frac{1}{2} + \Pr(\mathbf{A} \wedge \mathbf{H}),$$

that is $\Pr(\mathbf{A} \wedge \mathbf{H}) = \mathrm{Adv}_{\mathrm{P1}}^{\mathrm{eCK}}(\mathcal{A})$. Henceforth, the event $(\mathbf{A} \wedge \mathbf{H})$ is denoted as $\mathbf{A}^*$.

Note 1. Let $\mathcal{B}$ be an algorithm against a GDH challenger. $\mathcal{B}$ receives $L = g^\ell, W = g^w$ as the GDH challenge and $\mathcal{B}$ has access to a DDH oracle, which outputs 1 if the input is a tuple of $(g^\alpha, g^\beta, g^{\alpha\beta})$. $\Omega : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$ is a random function known only to $\mathcal{B}$, such that $\Omega(\Phi, \Theta) = \Omega(\Theta, \Phi)$ for all $\Phi, \Theta \in \mathbb{G}$. $\mathcal{B}$ will use $\Omega(\Phi, \Theta)$ as $\mathrm{CDH}(\Phi, \Theta)$ in situations where $\mathcal{B}$ does not know $\log_g \Phi$ and $\log_g \Theta$. Except with negligible probability, $\mathcal{A}$ will not recognize that $\Omega(\Phi, \Theta)$ is being used as $\mathrm{CDH}(\Phi, \Theta)$.

We construct the algorithm $\mathcal{B}$ using $\mathcal{A}$ as a sub-routine. $\mathcal{B}$ receives $L = g^\ell, W = g^w$ as the GDH challenge. We consider the following mutually exclusive events, under two main cases:

1. A partner to the test session exists: the adversary is allowed to corrupt both principals or reveal ephemeral keys from both sessions of the test session.
   (a) Adversary corrupts both the owner and partner principals to the test session - Event $\mathbf{E_{1a}}$
   (b) Adversary corrupts neither owner or nor partner principal to the test session - Event $\mathbf{E_{1b}}$

(c) Adversary corrupts the owner to the test session, but does not corrupt the partner to the test session - Event $\mathbf{E_{1c}}$

(d) Adversary corrupts the partner to the test session, but does not corrupt the owner to the test session - Event $\mathbf{E_{1d}}$

2. A partner to the test session does not exist: the adversary is not allowed to corrupt the intended partner principal to the test session.

(a) Adversary corrupts the owner to the test session - Event $\mathbf{E_{2a}}$

(b) Adversary does not corrupt the owner to the test session - Event $\mathbf{E_{2b}}$

In any other situation the test session is no longer fresh. If event $\mathbf{A}^*$ happens with non-negligible probability at least one of the following event should happen with non-negligible probability.

$$[(\mathbf{E_{1a}} \wedge \mathbf{A}^*), (\mathbf{E_{1b}} \wedge \mathbf{A}^*), (\mathbf{E_{1c}} \wedge \mathbf{A}^*), (\mathbf{E_{1d}} \wedge \mathbf{A}^*), (\mathbf{E_{2a}} \wedge \mathbf{A}^*), (\mathbf{E_{2b}} \wedge \mathbf{A}^*)]$$

Hence,

$$\begin{aligned}\mathrm{Adv}_{\mathrm{P1}}^{\mathrm{eCK}} \leq \max \Big(& \Pr(\mathbf{E_{1a}} \wedge \mathbf{A}^*), \Pr(\mathbf{E_{1b}} \wedge \mathbf{A}^*), \Pr(\mathbf{E_{1c}} \wedge \mathbf{A}^*), \\ & \Pr(\mathbf{E_{1d}} \wedge \mathbf{A}^*), \Pr(\mathbf{E_{2a}} \wedge \mathbf{A}^*), \Pr(\mathbf{E_{2b}} \wedge \mathbf{A}^*) \Big) \ .\end{aligned}$$

Adversary corrupts both the owner and partner principals to the test session - Event $\mathbf{E_{1a}}$

Event $(\mathbf{E_{1a}} \wedge \mathbf{A}^*)$: setup.

$\mathcal{B}$ establishes N_P number of honest parties to which $\mathcal{B}$ assigns long-term secret/public key pairs. For each honest party $\mathcal{B}$ maintains at most N_S number of sessions. $\mathcal{B}$ chooses two distinct random principals $U^*, V^* \xleftarrow{\$} \{U_1, \ldots, U_{N_P}\}$ and two random numbers $s^*, t^* \xleftarrow{\$} \{1, \ldots, N_S\}$. $\mathcal{B}$ guesses the session $\Pi^{s^*}_{U^*,V^*}$ as the target session and the session $\Pi^{t^*}_{V^*,U^*}$ as the partner to the target session. For the rest of this event consider A as the long-term public key of U^* and B as the long-term public key of V^*.

Event $(\mathbf{E_{1a}} \wedge \mathbf{A}^*)$: **simulation.**

- `Send`: $\mathcal{B}$ uses $X = L = g^{\ell}$ as the ephemeral public key of $\Pi^{s^*}_{U^*,V^*}$ and $Y = W = g^{w}$ as the ephemeral public key of $\Pi^{t^*}_{V^*,U^*}$. Note that $\mathcal{B}$ does not possess the ephemeral secret keys of sessions $\Pi^{s^*}_{U^*,V^*}$ and $\Pi^{t^*}_{V^*,U^*}$.
- `Corrupt`: $\mathcal{B}$ answers all `Corrupt` queries faithfully.
- `EphemeralKeyReveal`: If `EphemeralKeyReveal` query to the session $\Pi^{s^*}_{U^*,V^*}$ or $\Pi^{t^*}_{V^*,U^*}$ is asked, $\mathcal{B}$ aborts the simulation. Otherwise $\mathcal{B}$ answers all `EphemeralKeyReveal` queries faithfully.
- `SessionKeyReveal`: $\mathcal{B}$ answers all `SessionKeyReveal` queries faithfully.
- H($pos_1, pos_2, pos_3, pos_4, initiator, I, responder, J$): $\mathcal{B}$ simulates the random oracle H in the usual way. If $\mathcal{A}$ asks a H query such that $pos_4 = \mathrm{CDH}(L, W)$, $\mathcal{B}$ aborts the game and answers the GDH challenge ($\mathcal{B}$ can find whether $pos_4 = \mathrm{CDH}(L, W)$ or not by using the DDH oracle).
- `Test`: If the `Test` query is not asked to $\Pi^{s^*}_{U^*,V^*}$ with partner $\Pi^{t^*}_{V^*,U^*}$, $\mathcal{B}$ aborts the simulation. Otherwise, $\mathcal{B}$ obtains $K \leftarrow \mathrm{H}(\mathrm{CDH}(A, B), \mathrm{CDH}(B, L), \mathrm{CDH}(A, W), \Omega(L, W), U^*, L, V^*, W)$ (considering U^* is the initiator, otherwise exchange the positions of (CDH(B, L), CDH(A, W)), (U^*, V^*) and (L, W) respectively), and uses K as the real session key.

Event $(\mathbf{E_{1a}} \wedge \mathbf{A}^*)$: **analysis.**

We explained above that it is possible to perfectly simulate the answers to all the adversarial queries. The probability that $\mathcal{A}$ selects the sessions $\Pi^{s^*}_{U^*,V^*}$ and $\Pi^{t^*}_{V^*,U^*}$ as the test session and its partner is at least $\frac{1}{N_P^2 N_S^2}$, and the probability of event $(\mathbf{E_{1a}} \wedge \mathbf{A}^*)$ is non-negligible. The simulation of the view of eCK challenger to the adversary $\mathcal{A}$ is perfect except with negligible probability. According to the event $\mathbf{A}^*$, $\mathcal{A}$ queries the random oracle H with $(\mathrm{CDH}(A, B), \mathrm{CDH}(B, L), \mathrm{CDH}(A, W), \mathrm{CDH}(L, W), U^*, L, V^*, W)$. Hence, $\mathcal{B}$ can answer the GDH challenge with probability,

$$\mathrm{Pr}^{\mathrm{GDH}}_{g,q}(\mathcal{B}) \geq \frac{\mathrm{Pr}\left(\mathbf{E_{1a}} \wedge \mathbf{A}^*\right)}{N_P^2 N_S^2} \ .$$

Thus,

$$\mathrm{Pr}\left(\mathbf{E_{1a}} \wedge \mathbf{A}^*\right) \leq N_P^2 N_S^2 \Big(\mathrm{Pr}^{\mathrm{GDH}}_{g,q}(\mathcal{B})\Big) \ . \tag{3.1}$$

Adversary corrupts neither owner or nor partner principal to the test session - Event $\mathbf{E_{1b}}$

Event $(\mathbf{E_{1b}} \wedge \mathbf{A}^*)$: setup.

$\mathcal{B}$ establishes N_P number of honest parties. For each honest party $\mathcal{B}$ maintains at most N_S number of sessions. $\mathcal{B}$ chooses two distinct random principals $U^*, V^* \xleftarrow{\$} \{U_1, \ldots, U_{N_P}\}$. $\mathcal{B}$ sets $A = L = g^\ell$ as the long-term public key of U^* and $B = W = g^w$ as the long-term public key of V^*. For the rest of the parties $\mathcal{B}$ sets long-term secret/public key pairs according to the protocol specification. Note that $\mathcal{B}$ does not possess the long-term secret keys of U^* and V^*.

Event $(\mathbf{E_{1b}} \wedge \mathbf{A}^*)$: simulation.

- `Send`: On behalf of honest protocol principals $\mathcal{B}$ selects ephemeral secret/public key pairs according to the protocol specification.
- `Corrupt`: If `Corrupt` query to U^* or V^* is asked, $\mathcal{B}$ aborts the simulation. Otherwise $\mathcal{B}$ answers all `Corrupt` queries faithfully.
- `EphemeralKeyReveal`: $\mathcal{B}$ answers all `EphemeralKeyReveal` queries faithfully.
- `SessionKeyReveal`: $\mathcal{B}$ answers all `SessionKeyReveal` queries as follows:
  1. If $\mathcal{A}$ asks a `SessionKeyReveal` query to a session where both U^* and V^* are involved (Π_{U^*,V^*} or Π_{V^*,U^*}), $\mathcal{B}$ uses the function Ω to compute a value to replace $\mathrm{CDH}(L, W)$ as $\Omega(L, W)$.
  2. If $\mathcal{A}$ asks a `SessionKeyReveal` query to a session which is owned by principal U^* (or V^*) and does not have a partner (where $\mathcal{A}$ sends E having come from the partner and $\mathcal{B}$ does not possess the secret key corresponding to E), $\mathcal{B}$ uses the function Ω to compute a value to replace $\mathrm{CDH}(L, E)$ (or $\mathrm{CDH}(W, E)$) as $\Omega(L, E)$ (or $\Omega(W, E)$).
  3. For any other `SessionKeyReveal` query $\mathcal{B}$ can easily compute the four corresponding Diffie-Hellman exponentiations, because $\mathcal{B}$ knows the long-term secret keys of other principals and ephemeral secret keys of partnered sessions. Then $\mathcal{B}$ queries the random oracle H with the corresponding values and answers the `SessionKeyReveal` query.

Since $\mathcal{B}$ uses the function Ω to compute values to replace Diffie-Hellman exponentiations in places where it does not possess both long-term and ephemeral secret keys, $\mathcal{B}$ can maintain consistency when querying the random oracle H.

- H($pos_1, pos_2, pos_3, pos_4, initiator, I, responder, J$):

  1. If *initiator, responder* $\notin \{U^*, V^*\}$, $\mathcal{B}$ simulates the random oracle H in the usual way.
  2. If *initiator* $= U^*$, $\mathcal{B}$ checks the random oracle for a previously asked query, matching with the current one. If a match is found, $\mathcal{B}$ answers with the corresponding random-oracle value. Otherwise:
     - If $\mathcal{B}$ found all the positions except position 3 of a previously asked random oracle query respectively matching with the positions of the current random oracle query, $\mathcal{B}$ queries the DDH oracle with (L, J, pos_3). If the output is 1 and position 3 of the previously asked random oracle query equals to $\Omega(L, J)$, $\mathcal{B}$ answers with the corresponding random-oracle value in the table. Else $\mathcal{B}$ answers with a random value and stores the query and the answer in the random oracle table.
     - Else, $\mathcal{B}$ answers with a random value and stores the query and the answer in the random oracle table.

     Similarly when *initiator* $= V^*$ (but with W replacing L).
  3. If *responder* $= V^*$, $\mathcal{B}$ checks the random oracle for a previously asked query, matching with the current one. If a match is found, $\mathcal{B}$ answers with the corresponding random-oracle value. Otherwise:
     - If $\mathcal{B}$ found all the positions except position 2 of a previously asked random oracle query respectively matching with the positions of the current random oracle query, $\mathcal{B}$ queries the DDH oracle with (W, I, pos_2). If the output is 1 and position 2 of the previously asked random oracle query equals to $\Omega(W, I)$, $\mathcal{B}$ answers with the corresponding random-oracle value in the table. Else $\mathcal{B}$ answers with a random value and stores the query and the answer in the random oracle table.

– Else, $\mathcal{B}$ answers with a random value and stores the query and the answer in the random oracle table.

Similarly when $responder = U^*$ (but with L replacing W).

4. If $initiator \in \{U^*, V^*\}$ and $responder \in \{U^*, V^*\}$ and $\mathcal{A}$ asks a H query such that $pos_1 = \text{CDH}(L, W)$, $\mathcal{B}$ aborts the game and answers the GDH challenge ($\mathcal{B}$ can find whether $pos_1 = \text{CDH}(L, W)$ or not by using the DDH oracle).

- `Test`: If the `Test` query is not asked to a session where both U^* and V^* are involved ($\Pi^{\cdot}_{U^*,V^*}$ or $\Pi^{\cdot}_{V^*,U^*}$), $\mathcal{B}$ aborts the simulation. Otherwise, $\mathcal{B}$ obtains $K \leftarrow \text{H}(\Omega(L, W), \text{CDH}(W, X), \text{CDH}(L, Y), \text{CDH}(X, Y), U^*, X, V^*, Y)$ (considering U^* is the initiator, otherwise exchange the positions of ($\text{CDH}(W, X)$, $\text{CDH}(L, Y)$), (U^*, V^*) and (X, Y) respectively), and uses K as the real session key.

Event $(\mathbf{E_{1b}} \wedge \mathbf{A}^*)$: analysis.

We explained above that it is possible to perfectly simulate the answers to all the adversarial queries. The probability that $\mathcal{A}$ selects a session where both U^* and V^* are involved ($\Pi^{\cdot}_{U^*,V^*}$ or $\Pi^{\cdot}_{V^*,U^*}$) as the test session is at least $\frac{1}{N_P^2}$, and the probability of event $(\mathbf{E_{1b}} \wedge \mathbf{A}^*)$ is non-negligible. According to the event $\mathbf{A}^*$, $\mathcal{A}$ queries the random oracle H with $(\text{CDH}(L, W), \text{CDH}(W, X), \text{CDH}(L, Y), \text{CDH}(X, Y), U^*, X, V^*, Y)$. Hence, $\mathcal{B}$ can answer the GDH challenge with the probability,

$$\text{Pr}^{\text{GDH}}_{g,q}(\mathcal{B}) \geq \frac{\text{Pr}\left(\mathbf{E_{1b}} \wedge \mathbf{A}^*\right)}{N_P^2} \ .$$

Thus,

$$\text{Pr}\left(\mathbf{E_{1b}} \wedge \mathbf{A}^*\right) \leq N_P^2 \Big(\text{Pr}^{\text{GDH}}_{g,q}(\mathcal{B})\Big) \ . \tag{3.2}$$

Adversary corrupts the owner to the test session, but does not corrupt the partner to the test session - Event $\mathbf{E_{1c}}$

Event $(\mathbf{E_{1c}} \wedge \mathbf{A}^*)$: setup.

$\mathcal{B}$ establishes N_P number of honest parties. For each honest party $\mathcal{B}$ maintains at most N_S number of sessions. $\mathcal{B}$ chooses two distinct random principals $U^*, V^* \xleftarrow{\$} \{U_1, \ldots, U_{N_P}\}$. $\mathcal{B}$ sets $B = W = g^w$ as the long-term public key of V^*.

For the rest of the parties $\mathcal{B}$ sets long-term secret/public key pairs according to the protocol specification. Note that $\mathcal{B}$ does not possess the long-term secret keys of V^*. $\mathcal{B}$ chooses a random number $s^* \xleftarrow{\$} \{1, \ldots, N_S\}$ and guesses the session $\Pi^{s^*}_{U^*,V^*}$ as the target session.

Event $(\mathbf{E_{1c}} \wedge \mathbf{A}^*)$: simulation.

In the following description we assume that U^* is the initiator of the target session. The same analysis can also be applied to the case where U^* is the responder by adjusting the description as follows: In any place where we consider the random oracle query $\mathrm{H}(\mathrm{CDH}(A,B), \Omega(W,L), \mathrm{CDH}(A,Y), \mathrm{CDH}(X,Y), U^*, L, V^*, Y)$, exchange the positions of $(\Omega(W,L),\ \mathrm{CDH}(A,Y))$, $(U^*,\ V^*)$ and (X,Y) respectively. Particularly, this change is relevant in the simulation of the `Test` query and in the simulation of point 4 of the H query (consider pos_3 instead of pos_2).

- `Send`: $\mathcal{B}$ uses $X = L = g^{\ell}$ as the ephemeral public key of $\Pi^{s^*}_{U^*,V^*}$. Note that $\mathcal{B}$ does not possess the ephemeral secret key of session $\Pi^{s^*}_{U^*,V^*}$.
- `Corrupt`: If `Corrupt` query to V^* is asked, $\mathcal{B}$ aborts the simulation. Otherwise $\mathcal{B}$ answers all `Corrupt` queries faithfully.
- `EphemeralKeyReveal`: If `EphemeralKeyReveal` query to $\Pi^{s^*}_{U^*,V^*}$ is asked, $\mathcal{B}$ aborts the simulation. Otherwise $\mathcal{B}$ answers all `EphemeralKeyReveal` queries faithfully.
- `SessionKeyReveal`: $\mathcal{B}$ answers all `SessionKeyReveal` queries as follows:
  1. If $\mathcal{A}$ asks a `SessionKeyReveal` query to a session which is owned by principal V^* and does not have a partner (where $\mathcal{A}$ sends E having come from the partner and $\mathcal{B}$ does not possess the secret key corresponding to E), $\mathcal{B}$ uses the function Ω to compute a value to replace $\mathrm{CDH}(W,E)$ as $\Omega(W,E)$.
  2. For any other `SessionKeyReveal` query $\mathcal{B}$ can easily compute the four corresponding Diffie-Hellman exponentiations, because $\mathcal{B}$ knows the long-term secret keys of other principals and ephemeral secret keys of partnered sessions. Then $\mathcal{B}$ queries the random oracle H with the corresponding values and answers the `SessionKeyReveal` query.

Since $\mathcal{B}$ uses the function Ω to compute values to replace Diffie-Hellman exponentiations in places where it does not possess both long-term and ephemeral secret keys, $\mathcal{B}$ can maintain consistency when querying the random oracle H.

- H($pos_1, pos_2, pos_3, pos_4, initiator, I, responder, J$):

  1. If *initiator, responder* $\notin \{V^*\}$, $\mathcal{B}$ simulates the random oracle H in the usual way.
  2. If *initiator* $= V^*$, $\mathcal{B}$ checks the random oracle for a previously asked query, matching with the current one. If a match is found, $\mathcal{B}$ answers with the corresponding random-oracle value. Otherwise:
     - If $\mathcal{B}$ found all the positions except position 3 of a previously asked random oracle query respectively matching with the positions of the current random oracle query, $\mathcal{B}$ queries the DDH oracle with (W, J, pos_3). If the output is 1 and position 3 of the previously asked random oracle query equals to $\Omega(W, J)$, $\mathcal{B}$ answers with the corresponding random-oracle value in the table. Else $\mathcal{B}$ answers with a random value and stores the query and the answer in the random oracle table.
     - Else, $\mathcal{B}$ answers with a random value and stores the query and the answer in the random oracle table.
  3. If *responder* $= V^*$, $\mathcal{B}$ checks the random oracle for a previously asked query, matching with the current one. If a is match found, $\mathcal{B}$ answers with the corresponding random-oracle value. Otherwise:
     - If $\mathcal{B}$ found all the positions except position 2 of a previously asked random oracle query respectively matching with the positions of the current random oracle query, $\mathcal{B}$ queries the DDH oracle with (W, I, pos_2). If the output is 1 and position 2 of the previously asked random oracle query equals to $\Omega(W, I)$, $\mathcal{B}$ answers with the corresponding random-oracle value in the table. Else $\mathcal{B}$ answers with a random value and stores the query and the answer in the random oracle table.
     - Else, $\mathcal{B}$ answers with a random value and stores the query and the answer in the random oracle table.

4. If $\mathcal{A}$ asks a H query such that $pos_2 = \text{CDH}(W, L)$, $\mathcal{B}$ aborts the game and answers the GDH challenge ($\mathcal{B}$ can find whether $pos_2 = \text{CDH}(W, L)$ or not by using the DDH oracle).

- `Test`: If the `Test` query is not asked to $\Pi^{s^*}_{U^*,V^*}$, $\mathcal{B}$ aborts the simulation. Otherwise, $\mathcal{B}$ obtains $K \leftarrow \text{H}(\text{CDH}(A, W), \Omega(W, L), \text{CDH}(L, Y), \text{CDH}(X, Y), U^*, L, V^*, Y)$, and uses K as the real session key.

Event $(\mathbf{E_{1c}} \wedge \mathbf{A}^*)$: analysis.

We explained above that it is possible to perfectly simulate the answers to all the adversarial queries. The probability that $\mathcal{A}$ selects the session $\Pi^{s^*}_{U^*,V^*}$ as the test session is at least $\frac{1}{N_P^2 N_S}$, and the probability of event $(\mathbf{E_{1c}} \wedge \mathbf{A}^*)$ is non-negligible. According to the event $\mathbf{A}^*$, $\mathcal{A}$ queries the random oracle H with $(\text{CDH}(A, W), \text{CDH}(W, L), \text{CDH}(A, Y), \text{CDH}(L, Y), U^*, L, V^*, Y)$. Hence, $\mathcal{B}$ can answer the GDH challenge with the probability,

$$\text{Pr}^{\text{GDH}}_{g,q}(\mathcal{B}) \geq \frac{\text{Pr}\left(\mathbf{E_{1c}} \wedge \mathbf{A}^*\right)}{N_P^2 N_S} \ .$$

Thus,

$$\text{Pr}\left(\mathbf{E_{1c}} \wedge \mathbf{A}^*\right) \leq N_P^2 N_S \Big(\text{Pr}^{\text{GDH}}_{g,q}(\mathcal{B})\Big) \ . \tag{3.3}$$

Adversary corrupts the partner to the test session, but does not corrupt the owner to the test session - Event $\mathbf{E_{1d}}$

Event $(\mathbf{E_{1d}} \wedge \mathbf{A}^*)$: setup.

$\mathcal{B}$ establishes N_P number of honest parties. For each honest party $\mathcal{B}$ maintains at most N_S number of sessions. $\mathcal{B}$ chooses two distinct random principals $U^*, V^* \xleftarrow{\$} \{U_1, \ldots, U_{N_P}\}$. $\mathcal{B}$ sets $A = W = g^w$ as the long-term public key of U^*. For the rest of the parties $\mathcal{B}$ sets long-term secret/public key pairs according to the protocol specification. Note that $\mathcal{B}$ does not possess the long-term secret keys of U^*. $\mathcal{B}$ chooses a random number $t^* \xleftarrow{\$} \{1, \ldots, N_S\}$ and guesses the session $\Pi^{t^*}_{V^*,U^*}$ as the partner session to the target session.

Event $(\mathbf{E_{1d}} \wedge \mathbf{A}^*)$: simulation.

In the following description we assume that U^* is the initiator of the target session. The same analysis also be applied to the case where U^* is the responder by

adjusting the description as follows: In any place where we consider the random oracle query $\mathrm{H}(\mathrm{CDH}(W,B),\mathrm{CDH}(B,X),\Omega(W,L),\mathrm{CDH}(X,L),U^*,X,V^*,L)$, exchange the positions of $(\mathrm{CDH}(B,X),\ \Omega(A,Y))$, $(U^*,\ V^*)$ and (X,Y) respectively. Particularly, this change is relevant in the simulation of the `Test` query and in the simulation of the point 4 of the H query checks for $pos_2 = \mathrm{CDH}(W,L)$.

- `Send`: $\mathcal{B}$ uses $Y = L = g^{\ell}$ as the ephemeral public key of $\Pi^{t^*}_{V^*,U^*}$. Note that $\mathcal{B}$ does not posses the ephemeral secret key of session $\Pi^{t^*}_{V^*,U^*}$.

- `Corrupt`: If `Corrupt` query to U^* is asked, $\mathcal{B}$ aborts the simulation. Otherwise $\mathcal{B}$ answers all `Corrupt` queries faithfully.

- `EphemeralKeyReveal`: If `EphemeralKeyReveal` query to partner to $\Pi^{t^*}_{V^*,U^*}$ is asked, $\mathcal{B}$ aborts the simulation. Otherwise answers all `EphemeralKeyReveal` queries faithfully.

- `SessionKeyReveal`: $\mathcal{B}$ answers all `SessionKeyReveal` queries as follows:

  1. If $\mathcal{A}$ asks a `SessionKeyReveal` query to a session which is owned by principal U^* and does not have a partner (where $\mathcal{A}$ sends E having come from the partner and $\mathcal{B}$ does not possess the secret key corresponding to E), $\mathcal{B}$ uses the function Ω to compute a value to replace $\mathrm{CDH}(W,E)$ as $\Omega(W,E)$.
  2. For any other `SessionKeyReveal` query $\mathcal{B}$ can easily compute the four corresponding Diffie-Hellman exponentiations, because $\mathcal{B}$ knows the long-term secret keys of other principals and ephemeral secret keys of partnered sessions. Then $\mathcal{B}$ queries the random oracle H with the corresponding values and answers the `SessionKeyReveal` query.

  Since $\mathcal{B}$ uses the function Ω to compute values to replace Diffie-Hellman exponentiations in places where it does not possess both long-term and ephemeral secret keys, $\mathcal{B}$ can maintain consistency when querying the random oracle H.

- $\mathrm{H}(pos_1, pos_2, pos_3, pos_4, \mathit{initiator}, I, \mathit{responder}, J)$:

  1. If $\mathit{initiator},\ \mathit{responder} \notin \{U^*\}$, $\mathcal{B}$ simulates the random oracle H in the usual way.

2. If *initiator* $= U^*$, $\mathcal{B}$ checks the random oracle for a previously asked query, matching with the current one. If a match is found, $\mathcal{B}$ answers with the corresponding random-oracle value. Otherwise:
   - If $\mathcal{B}$ found all the positions except position 3 of a previously asked random oracle query respectively matching with the positions of the current random oracle query, $\mathcal{B}$ queries the DDH oracle with (W, J, pos_3). If the output is 1 and position 3 of the previously asked random oracle query equals to $\Omega(W, J)$,$\mathcal{B}$ answers with the corresponding random-oracle value in the table. Else $\mathcal{B}$ answers with a random value and stores the query and the answer in the random oracle table.
   - Else, $\mathcal{B}$ answers with a random value and stores the query and the answer in the random oracle table.
3. If *responder* $= U^*$, $\mathcal{B}$ checks the random oracle for a previously asked query, matching with the current one. If a match is found, $\mathcal{B}$ answers with the corresponding random-oracle value. Otherwise:
   - If $\mathcal{B}$ found all the positions except position 2 of a previously asked random oracle query respectively matching with the positions of the current random oracle query, $\mathcal{B}$ queries the DDH oracle with (W, I, pos_2). If the output is 1 and position 2 of the previously asked random oracle query equals to $\Omega(W, I)$, $\mathcal{B}$ answers with the corresponding random-oracle value in the table. Else $\mathcal{B}$ answers with a random value and stores the query and the answer in the random oracle table.
   - Else, $\mathcal{B}$ answers with a random value and stores the query and the answer in the random oracle table.
4. If $\mathcal{A}$ asks a H query such that $pos_3 = \mathrm{CDH}(W, L)$, $\mathcal{B}$ aborts the game and answers the GDH challenge ($\mathcal{B}$ can find whether $pos_3 = \mathrm{CDH}(W, L)$ or not by using the DDH oracle.)

- `Test`: If the `Test` query is not asked to the partner of $\Pi^{t^*}_{V^*,U^*}$, $\mathcal{B}$ aborts the simulation. Otherwise, $\mathcal{B}$ obtains $K \leftarrow \mathrm{H}(\mathrm{CDH}(W, B), \mathrm{CDH}(B, X), \Omega(W, L), \mathrm{CDH}(X, L), U^*, X, V^*, L)$, and uses K as the real session key.

Event $(\mathbf{E_{1d}} \wedge \mathbf{A}^*)$: analysis.

We explained above that it is possible to perfectly simulate the answers to all the adversarial queries. The probability that $\mathcal{A}$ selects the partner to the session $\Pi^{t^*}_{V^*,U^*}$ as the test session is at least $\frac{1}{N_P^2 N_S}$, and the probability of event $(\mathbf{E_{1d}} \wedge \mathbf{A}^*)$ is non-negligible. According to the event $\mathbf{A}^*$, $\mathcal{A}$ queries the random oracle H with $(\mathrm{CDH}(W,B), \mathrm{CDH}(B,X), \mathrm{CDH}(W,L), \mathrm{CDH}(X,Y), U^*, X, V^*, L)$. Hence, $\mathcal{B}$ can answer the GDH challenge with the probability,

$$\Pr^{\mathrm{GDH}}_{g,q}(\mathcal{B}) \geq \frac{\Pr\left(\mathbf{E_{1d}} \wedge \mathbf{A}^*\right)}{N_P^2 N_S} \ .$$

Thus,

$$\Pr\left(\mathbf{E_{1d}} \wedge \mathbf{A}^*\right) \leq N_P^2 N_S \Big(\Pr^{\mathrm{GDH}}_{g,q}(\mathcal{B}) \Big) \ . \tag{3.4}$$

Adversary corrupts the owner to the test session - Event $\mathbf{E_{2a}}$

Event $(\mathbf{E_{2a}} \wedge \mathbf{A}^*)$: setup.

$\mathcal{B}$ establishes N_P number of honest parties. For each honest party $\mathcal{B}$ maintains at most N_S number of sessions. $\mathcal{B}$ chooses two distinct random principals $U^*, V^* \xleftarrow{\$} \{U_1, \ldots, U_{N_P}\}$. $\mathcal{B}$ sets $B = W = g^w$ as the long-term public key of V^*. For the rest of the parties $\mathcal{B}$ sets long-term secret/public key pairs according to the protocol specification. Note that $\mathcal{B}$ does not possess the long-term secret keys of V^*. $\mathcal{B}$ chooses a random number $s^* \xleftarrow{\$} \{1, \ldots, N_S\}$ and guesses the session $\Pi^{s^*}_{U^*,V^*}$ as the target session.

Event $(\mathbf{E_{2a}} \wedge \mathbf{A}^*)$: simulation.

$\mathcal{B}$ uses $X = L = g^\ell$ as the ephemeral public key of $\Pi^{s^*}_{U^*,V^*}$. Note that $\mathcal{B}$ does not posses the ephemeral secret key of session $\Pi^{s^*}_{U^*,V^*}$. This simulation is same as the simulation of Event $\mathbf{E_{1c}} \wedge \mathbf{A}^*$.

Event $(\mathbf{E_{2a}} \wedge \mathbf{A}^*)$: analysis.

We explained above that it is possible to perfectly simulate the answers to all the adversarial queries. The probability that $\mathcal{A}$ selects the session $\Pi^{s^*}_{U^*,V^*}$ as the test session is at least $\frac{1}{N_P^2 N_S}$, and the probability of event $(\mathbf{E_{2a}} \wedge \mathbf{A}^*)$ is non-negligible. According to the event $\mathbf{A}^*$, $\mathcal{A}$ queries the random oracle H with

$(\text{CDH}(A, W), \text{CDH}(W, L), \text{CDH}(A, Y), \text{CDH}(L, Y), U^*, L, V^*, Y)$. Hence, $\mathcal{B}$ can answer the GDH challenge with the probability,

$$\Pr_{g,q}^{\text{GDH}}(\mathcal{B}) \geq \frac{\Pr\left(\mathbf{E_{2a}} \wedge \mathbf{A}^*\right)}{N_P^2 N_S} \ .$$

Thus,

$$\Pr\left(\mathbf{E_{2a}} \wedge \mathbf{A}^*\right) \leq N_P^2 N_S \Big(\Pr_{g,q}^{\text{GDH}}(\mathcal{B})\Big) \ . \tag{3.5}$$

Adversary does not corrupt the owner to the test session - Event $\mathbf{E_{2b}}$

Event $(\mathbf{E_{2b}} \wedge \mathbf{A}^*)$: setup.

$\mathcal{B}$ establishes N_P number of honest parties. For each honest party $\mathcal{B}$ maintains at most N_S number of sessions. $\mathcal{B}$ chooses two distinct random principals $U^*, V^* \xleftarrow{\$} \{U_1, \ldots, U_{N_P}\}$. $\mathcal{B}$ sets $A = L = g^\ell$ as the long-term public key of U^* and $B = W = g^w$ as the long-term public key of V^*. For the rest of the parties $\mathcal{B}$ sets long-term secret/public key pairs according to the protocol specification. Note that $\mathcal{B}$ does not possess the long-term secret keys of U^* and V^*.

Event $(\mathbf{E_{2b}} \wedge \mathbf{A}^*)$: simulation.

This simulation is same as the simulation of Event $\mathbf{E_{1b}} \wedge \mathbf{A}^*$.

Event $(\mathbf{E_{2b}} \wedge \mathbf{A}^*)$: analysis.

We explained above that it is possible to perfectly simulate the answers to all the adversarial queries. The probability that $\mathcal{A}$ selects an oracle where both U^* and V^* involve ($\Pi^{\cdot}_{U^*,V^*}$ or $\Pi^{\cdot}_{V^*,U^*}$) as the test session is at least $\frac{1}{N_P^2}$, and the probability of event $(\mathbf{E_{1b}} \wedge \mathbf{A}^*)$ is non-negligible. According to the event $\mathbf{A}^*$, $\mathcal{A}$ queries the random oracle H with $(\text{CDH}(L, W), \text{CDH}(W, X), \text{CDH}(L, Y), \text{CDH}(X, Y), U^*, X, V^*, Y)$. Hence, $\mathcal{B}$ can answer the GDH challenge with the probability,

$$\Pr_{g,q}^{\text{GDH}}(\mathcal{B}) \geq \frac{\Pr\left(\mathbf{E_{2b}} \wedge \mathbf{A}^*\right)}{N_P^2} \ .$$

Thus,

$$\Pr\left(\mathbf{E_{2b}} \wedge \mathbf{A}^*\right) \leq N_P^2 \Big(\Pr_{g,q}^{\text{GDH}}(\mathcal{B})\Big) \ . \tag{3.6}$$

We know that,

$$\mathrm{Adv}_{\mathrm{P1}}^{\mathrm{eCK}} \leq \max\Big(\Pr(\mathbf{E_{1a}} \wedge \mathbf{A}^*), \Pr(\mathbf{E_{1b}} \wedge \mathbf{A}^*), \Pr(\mathbf{E_{1c}} \wedge \mathbf{A}^*), \Pr(\mathbf{E_{1d}} \wedge \mathbf{A}^*), \Pr(\mathbf{E_{2a}} \wedge \mathbf{A}^*), \Pr(\mathbf{E_{2b}} \wedge \mathbf{A}^*)\Big) \ .$$

Therefore we get that,

$$\mathrm{Adv}_{\mathrm{P1}}^{\mathrm{eCK}} \leq \max\Big(N_P^2 N_S^2\big(\mathrm{Pr}_{g,q}^{\mathrm{GDH}}(\mathcal{B})\big), N_P^2\big(\mathrm{Pr}_{g,q}^{\mathrm{GDH}}(\mathcal{B})\big), N_P^2 N_S\big(\mathrm{Pr}_{g,q}^{\mathrm{GDH}}(\mathcal{B})\big), N_P^2 N_S\big(\mathrm{Pr}_{g,q}^{\mathrm{GDH}}(\mathcal{B})\big), N_P^2 N_S\big(\mathrm{Pr}_{g,q}^{\mathrm{GDH}}(\mathcal{B})\big), N_P^2\big(\mathrm{Pr}_{g,q}^{\mathrm{GDH}}(\mathcal{B})\big)\Big) \ .$$

□

3.4 Comparison of Key Exchange Protocols

Table 3.5 summarizes the most significant properties of the protocols in this chapter. We can see the HMQV, the NAXOS, the CMQV and the P1 protocols have been proven using the random oracle assumption.

Property	MQV	HMQV	NAXOS	CMQV	P1
Security Proof	✗	CK	eCK	eCK	eCK
Random Oracle	✗	✔	✔	✔	✔
NAXOS Trick	✗	✗	✔	✔	✗
Diffie-Hellman Based	✔	✔	✔	✔	✔
KCI security	✗	✔	✔	✔	✔
UKS security	✗	✔	✔	✔	✔
Known key security	✗	✔	✔	✔	✔
Forward Secrecy (FS)	✗	weak FS	weak FS	weak FS	weak FS

Table 3.5: Comparison of key exchange protocols

Chapter 4

Continuous After-the-fact Leakage in Restricted-eCK Model

Contents

A number of security models for two-party authenticated key exchange (AKE) protocols have been developed over time. We discussed a few such security models in chapter 3. The main purpose of those security models is to provide a framework to analyze the security of key exchange protocols against a strong adversary, who is capable of learning certain secrets of the protocol principals. In

this chapter, we advance the modelling of AKE protocols by considering more granular, *continuous leakage* of long-term secrets of protocol participants: the adversary can adaptively request arbitrary leakage of long-term secrets even after the test session is activated *(after-the-fact)*, with limits on the amount of leakage per query but no bounds on the total leakage. Differently, in the bounded leakage model, the total amount of leakage is bounded. We present a security model supporting continuous leakage even when the adversary learns certain ephemeral secrets or session keys *(restricted* eCK *model)*. Moreover we give a generic construction of a two-pass leakage-resilient key exchange protocol that is secure in the model. The generic protocol can be instantiated with certain available public-key encryption schemes in the literature.

The generic protocol construction presented in our initial publication of this work [ABS14] was later found by Toorani [Too15] to be vulnerable to ephemeral key reveal attacks against an active adversary (legitimate partner does not exist for the target session): the adversary who picks the ephemeral secret key as the partner session to the target session, can trivially compute the session key of the target session, by revealing the ephemeral secret key of the target session from the owner of the target session. As a result, in this chapter, we revise the initial security model by restricting the active adversary to reveal the ephemeral secret key of the target session from the owner of the target session. Thus, in the revised security model, it is possible to prove the security of the generic protocol construction. Even after revising the initial model by enforcing this additional restriction, this model still allows the adversary to learn long-term, ephemeral secret keys of the target session, which is a more powerful adversarial capability than in many of the previous security models such as the Bellare–Rogaway [BR93] models and the Canetti–Krawczyk [CK01] model. In Chapter 5, we present a stronger security model and a proven secure generic protocol construction, overcoming the limitations we enforce in this chapter.

4.1 Introduction

Earlier key exchange security models, such as the Bellare–Rogaway [BR93], Canetti–Krawczyk [CK01], and extended Canetti–Krawczyk (eCK) [LLM07] models, aim to capture security against an adversary who can fully compromise some, but not all secret keys. For example, in the eCK model, a session key

should be secure even if the adversary has compromised either the long-term or ephemeral key at the client, and either the long-term or ephemeral key at the server, but not all of the values at one party. This is not a very granular form of leakage, and thus is not fully suitable for modelling side-channel attacks.

This motivates the development of leakage-resilient key exchange security models and protocols. Moriyama and Okamoto [MO11] created key exchange security models to analyze security of leakage-resilient key exchange protocols, using a variant of the eCK model. There are two central limitations in the Moriyama–Okamoto model as we pointed out in section 3.1.2. First, the total amount of leakage allowed in the Moriyama–Okamoto model is bounded. Second, the adversary cannot obtain any leakage information after the "test" session is activated. The former restriction is troublesome because, in practice, ongoing executions of a protocol may reveal a small amount of leakage each time, and we would like to provide security against this "continuous" leakage. The latter restriction is problematic because we would like to provide security of one session, even if some leakage happens in subsequent sessions.

The above limitations thus somewhat restrict the adversary in the Moriyama–Okamoto model. In this chapter, we aim to remove these two restrictions. We will present a model and a protocol that allow the adversary to adaptively obtain an unbounded amount of total *continuous leakage*, albeit with the restriction that the amount of leakage obtained in each leakage request is limited; this addresses the first restriction of previous models. Secondly, we allow the adversary to obtain leakage after the test session is activated, addressing the second restriction of the Moriyama–Okamoto model. The model we present in this chapter enforces restrictions on the freshness condition, in addition to the eCK-style freshness condition defined in the Moriyama–Okamoto model. We explain the additional restrictions on the freshness condition further in this chapter.

4.2 Continuous After-the-fact Leakage (CAFL) Model

In this section we introduce the continuous after-the-fact leakage model, (CAFL model), for key exchange. In the CAFL model, the adversary is allowed to adaptively obtain partial leakage on the long-term secret keys even after the test session is activated, as well as reveal session keys, long-term keys, and ephemeral

keys.

4.2.1 Modelling Leakage

In this key exchange security model we consider *continuous leakage of the long-term secret keys* of protocol principals, because long-term secret keys are not one-time secrets, but they last for multiple protocol sessions. Leakage of long-term secret key from one session affects the security of another session which uses the same long-term secret key. Considering side-channel attacks which can be mounted against key exchange protocols, the most realistic way to obtain the leakage information of long-term secret keys is from the protocol computations which use long-term secret keys. Hence, following the premise "only computation leaks information" [MR04], we have modeled the leakage to occur where computation takes place using secret keys. By issuing a `Send` query, the adversary will get a protocol message which is computed according to the normal protocol computations. Therefore, the instance of a `Send` query would be the appropriate instance to address the leakage which occurs due to a computation which uses a long-term secret key. Thus, sending an adversary-chosen leakage function, f, with the `Send` query would reflect the premise "only computation leaks information". We assume also that the leakage function f is an efficiently computable, adaptive leakage function.

Further, we assume that the amount of leakage of a secret key is bounded by a leakage parameter λ, per computation. The adversary is allowed to obtain leakage from many computations continuously. Hence, the overall leakage amount is unbounded.

4.2.2 Adversarial Powers

The adversary (a probabilistic algorithm) controls all interaction and communication between parties. In particular, the adversary initiates sessions at parties and delivers protocol messages; it can create, change, delete, or reorder messages. The adversary can also compromise certain short-term and long-term secrets. Notably, whenever the party performs an operation using its long-term key, the adversary obtains some leakage information about the long-term key.

The following query allows the adversary $\mathcal{A}$ to run the protocol, modelling normal communication.

- `Send`(U, V, s, m, f) query: The session $\Pi^s_{U,V}$, computes the next protocol message according to the protocol specification on receipt of m, and sends it to the adversary $\mathcal{A}$, along with the leakage $f(sk_U)$ as described in Section 4.2.1. $\mathcal{A}$ can also use this query to activate a new protocol instance as an initiator with blank m.

The following queries allow the adversary $\mathcal{A}$ to compromise certain session specific ephemeral secrets and long-term secrets from the protocol principals.

- `SessionKeyReveal`(U, V, s) query: $\mathcal{A}$ is given the session key of the session $\Pi^s_{U,V}$, if the session $\Pi^s_{U,V}$ is in the accepted state.

- `EphemeralKeyReveal`(U, V, s) query: $\mathcal{A}$ is given the ephemeral keys of the session $\Pi^s_{U,V}$.

- `Corrupt`(U) query: $\mathcal{A}$ is given the long-term secrets of the principal U. This query does not reveal any session keys or ephemeral keys to $\mathcal{A}$.

Remark 4.2.1 (`Corrupt` query vs Leakage queries)**.** By issuing a `Corrupt` query, the adversary gets the party's entire long-term secret key. Separately, by issuing leakage queries (using leakage function f embedded with the `Send` query) the adversary gets λ-bounded amount of leakage information about the long-term secret key. It may seem paradoxical to consider `Corrupt` and Leakage queries at the same time. But there are good reasons to consider both.

- A *non-leakage* version of CAFL model (`Send` query without f) addresses KCI attacks, because the adversary is allowed to corrupt the owner of the test session before the activation of the test session. In the CAFL model, we allow the adversary to obtain leakage from the partner of the test session, in addition to allowing the adversary to corrupt the owner of the test session.

- A *non-leakage* version of CAFL model (`Send` query without f) addresses partial weak forward secrecy, because the adversary is allowed to corrupt either of the protocol principals, but not both, after the test session is activated. In the CAFL model, we allow the adversary to obtain leakage from the uncorrupted principal, in addition to allowing the adversary to corrupt one of the protocol principals.

Hence, the CAFL model allows the adversary to obtain more information than a *non-leakage* version of CAFL model.

4.2.3 Defining Security

In this section we give formal definitions for partner sessions, freshness of a session and security in the CAFL model.

Definition 4.2.1 (Partner sessions in CAFL model)**.** As in the eCK model, two sessions $\Pi^{s}_{U,V}$ and $\Pi^{s'}_{U',V'}$ are said to be partners if:

1. $\Pi^{s}_{U,V}$ and $\Pi^{s'}_{U',V'}$ have computed session keys and
2. Sent messages from $\Pi^{s}_{U,V}$ = Received messages to $\Pi^{s'}_{U',V'}$ and
3. Sent messages from $\Pi^{s'}_{U',V'}$ = Received messages to $\Pi^{s}_{U,V}$ and
4. $U' = V$ and $V' = U$ and
5. If U is the initiator then V is the responder, or vice versa.

A protocol is said to be *correct* if two partner sessions compute identical session keys in the presence of a passive adversary.

Once the session $\Pi^{s}_{U,V}$ has accepted a session key, the adversary $\mathcal{A}$ attempts to distinguish it from a random session key, by asking the following query. The `Test` query is used to formalize the notion of the semantic security of a key exchange protocol.

- `Test`(U, V, s) query: When $\mathcal{A}$ asks the `Test` query, the session $\Pi^{s}_{U,V}$ first chooses a random bit $b \leftarrow \{0, 1\}$ and if $b = 1$ then the actual session key is returned to $\mathcal{A}$, otherwise a random string chosen from the same session key space is returned to $\mathcal{A}$. This query is only allowed to be asked once across all sessions.

We now define what it means for a session to be $\lambda-$CAFL-*fresh* in the CAFL model.

Definition 4.2.2 ($\lambda-$CAFL-freshness)**.** Let λ be the leakage bound per occurrence. A session $\Pi^{s}_{U,V}$ is said to be $\lambda-$CAFL-*fresh* if and only if:

1. The session $\Pi^{s}_{U,V}$ or its partner, $\Pi^{s'}_{V,U}$ (if it exists) has not been asked a `SessionKeyReveal`.
2. If the partner $\Pi^{s'}_{V,U}$ exists, none of the following combinations have been asked:

   (a) $\texttt{Corrupt}(U)$ and $\texttt{Corrupt}(V)$.

   (b) $\texttt{Corrupt}(U)$ and $\texttt{EphemeralKeyReveal}(U, V, s)$.

   (c) $\texttt{Corrupt}(V)$ and $\texttt{EphemeralKeyReveal}(V, U, s')$.

   (d) $\texttt{EphemeralKeyReveal}(U, V, s)$ and $\texttt{EphemeralKeyReveal}(V, U, s')$.

3. If the partner $\Pi^{s'}_{V,U}$ does not exist, none of the following combinations have been asked:

   (a) $\texttt{Corrupt}(V)$.

   (b) $\texttt{EphemeralKeyReveal}(U, V, s)$.

4. For each $\texttt{Send}(U, \cdot, \cdot, \cdot, f)$ query, the output of f is at most λ bits.

5. For each $\texttt{Send}(V, \cdot, \cdot, \cdot, f)$ query, the output of f is at most λ bits.

Limitations of λ − CAFL-freshness. When the adversary asks `Corrupt` and `EphemeralKeyReveal` queries, there are two `Corrupt`–`EphemeralKeyReveal` query combinations which trivially expose the session key of a session, in a scenario that a partner to that particular session exists.

1. $\texttt{Corrupt}(U)$ and $\texttt{EphemeralKeyReveal}(U, V, s)$.

2. $\texttt{Corrupt}(V)$ and $\texttt{EphemeralKeyReveal}(V, U, s')$.

As in the other models we have compared with [LLM07, MO11] we do not allow above combinations in the freshness condition, as they trivially expose the session key of sessions $\Pi^{s}_{U,V}$ and $\Pi^{s'}_{V,U}$. Differently, in the other models we have compared with, there are four `Corrupt`–`EphemeralKeyReveal` query combinations which do not trivially expose the session key a session, in a scenario that a partner to that particular session exists.

1. $\texttt{Corrupt}(U)$ and $\texttt{Corrupt}(V)$.

2. $\texttt{Corrupt}(U)$ and $\texttt{EphemeralKeyReveal}(V, U, s)$.

3. $\texttt{Corrupt}(V)$ and $\texttt{EphemeralKeyReveal}(U, V, s')$.

4. $\texttt{EphemeralKeyReveal}(V, U, s)$ and $\texttt{EphemeralKeyReveal}(U, V, s')$.

All the models we consider [LLM07, MO11] allow above combinations in the freshness condition, whereas our CAFL model does not allow the query combinations 1 and 4 in the freshness condition.

When the adversary asks `EphemeralKeyReveal` and `Corrupt` queries, there are two query combinations which trivially expose the session key of a session, in a scenario that a partner to that particular session does not exist.

1. `Corrupt`(V).
2. `Corrupt`(U) and `EphemeralKeyReveal`(U, V, s).

As in the other models we have compared with [LLM07, MO11] we do not allow above combinations in the freshness condition, as they trivially expose the session key of sessions $\Pi^{s}_{U,V}$ and $\Pi^{s'}_{V,U}$. Weakening that condition, our model does not allow following two query combinations in the freshness condition, when a partner to the test session does not exist.

1. `Corrupt`(V).
2. `EphemeralKeyReveal`(U, V, s). (instead of `EphemeralKeyReveal`(U, V, s) and `Corrupt`(U) as in other models)

Thus, the freshness of a non-leakage variant of the CAFL model (without conditions 4 and 5) is weaker than the eCK-freshness definition, because of the restriction enforced in the conditions (2)-a and (2)-d. Differently, the $\lambda -$ CAFL-freshness allows partial leakage of the long-term secret key of a protocol principal, even when the partner principal is corrupted or `EphemeralKeyReveal` query is asked to the partner session. In some sense that is stronger than the eCK-freshness definition, because according to the eCK-freshness, once `EphemeralKeyReveal` query have been asked to a session, revealing the long-term secret key of the partner is not allowed. Hence, although the freshness of a non-leakage variant of the CAFL model is weaker than the eCK-freshness in some sense, $\lambda -$ CAFL-freshness achieved an improvement over eCK-freshness by means of partial leakage of long-term secrets.

We justify why we introduce additional restrictions (restriction enforced in the conditions (2)-a and (2)-d) to the freshness condition of the CAFL model, more than in the freshness condition of the eCK model as follows: At this stage our aim is to construct a simple leakage-resilient two-pass key exchange protocol, using a leakage-resilient public-key encryption scheme, in which each of the principals

randomly chooses its ephemeral secret key, encrypts it with the public key of the intended partner principal, and sends the encrypted message to the intended partner principal. Defining eCK-style freshness makes it impossible to prove the security of our simple two-pass key exchange protocol, because corrupting both principals to the target session or revealing the ephemeral key from both sessions to the target session will trivially expose the session key. Therefore, we enforce additional restrictions to the $\lambda -$ CAFL-freshness condition, but allow the partial leakage of long-term secret keys, as our aim is to model the side-channel attacks. We will define a stronger eCK-style freshness condition for a leakage security model in chapter 5 and then construct a stronger leakage-resilient key exchange protocol.

Security of a key exchange protocol in the CAFL model is defined using the following security game, which is played by a probabilistic polynomial time adversary $\mathcal{A}$ against the protocol challenger.

- **Stage 0:** The challenger generates the keys by using the security parameter k.
- **Stage 1:** $\mathcal{A}$ may ask any of `Send`, `SessionKeyReveal`, `EphemeralKeyReveal` and `Corrupt` queries to any session at will.
- **Stage 2:** $\mathcal{A}$ chooses a $\lambda -$ CAFL-fresh session and asks a `Test` query.
- **Stage 3:** $\mathcal{A}$ may continue asking `Send`, `SessionKeyReveal`, `Corrupt` and `EphemeralKeyReveal` queries. $\mathcal{A}$ may not ask a query that violates the $\lambda -$ CAFL-freshness of the test session.
- **Stage 4:** Eventually, $\mathcal{A}$ outputs the bit $b' \in \{0, 1\}$ which is its guess of the value b on the test session. $\mathcal{A}$ wins if $b' = b$.

$\text{Succ}_{\mathcal{A}}$ is the event that $\mathcal{A}$ wins the above security game. The definition of security follows.

Definition 4.2.3 ($\lambda -$ CAFL-security)**.** A protocol π is said to be $\lambda -$ CAFL-secure if there is no probabilistic polynomial time algorithm $\mathcal{A}$ that can win the above game with non-negligible advantage. The advantage of an adversary $\mathcal{A}$ is defined as $\text{Adv}_{\pi}^{\lambda-\text{CAFL}}(\mathcal{A}) = |2\Pr(\text{Succ}_{\mathcal{A}}) - 1|$.

4.2.4 Practical Interpretation of Security of CAFL Model

We review the relationship between the CAFL model and real world attack scenarios.

- **Active adversarial capabilities:** `Send` queries address the powers of an active adversary who can control the message flow over the network. In the previous security models, this property is addressed by introducing the *send* query.

- **Side-channel attacks:** Leakage functions are embedded with the `Send` query. Thus, assuming that the leakage happens when computations take place in principals, a wide variety of side-channel attacks such as timing attacks, EM emission based attacks, power analysis attacks, which are based on *continuous leakage of long-term secrets* are addressed. This property is not addressed in the earlier security models such as the BR models, the CK model, the eCK model and the Moriyama-Okamoto model.

- **Cold boot attacks:** The CAFL model allows the adversary to reveal either the long-term secret key (`Corrupt` query) or the ephemeral secret key (`EphemeralKeyReveal` query) of the target session (same as in the eCK model and the Moriyama-Okamoto model). Thus these queries address the cold boot attacks to some extent, where the cold boot attacks reveal either (i) the long-term secret key or (ii) the ephemeral secret key of the target session. Note that the Moriyama-Okamoto model addresses the cold boot attacks by additionally covering the situation, where the attacker reveals (iii) the ephemeral secret key and part of the long-term secret key of the target session. Thus, the Moriyama-Okamoto model is more suitable to model cold boot attacks.

- **Malware attacks:** `EphemeralKeyReveal` queries cover the malware attacks which steal stored ephemeral keys, given that the long-term keys may be securely stored separately from the ephemeral keys in places such as smart cards or hardware security modules. Separately, `Corrupt` queries address malware attacks which steal the long-term secret keys of protocol principals. In the previous security models, this property is addressed by introducing the *ephemeral-key reveal, session-state reveal* and *corrupt* queries.

- **Weak random number generators:** Due to weak random number generators, the adversary may correctly determine the produced random number. `EphemeralKeyReveal` query addresses situations where the adversary can get the ephemeral secrets. In the previous security models, this property is addressed by introducing the *ephemeral-key reveal query* or the *session-state reveal* query.

- **Known key attacks:** `SessionKeyReveal` query covers the attacks which can be mounted by knowing past session keys. In the previous security models, this property is addressed by introducing the *session key reveal* query.

- **Key compromise impersonation attacks:** $\lambda -$ CAFL-freshness allows the adversary to corrupt the owner of the test session before the activation of the test session. Hence, the CAFL model security protects against the key compromise impersonation attacks. In the eCK model and the Moriyama-Okamoto model, this property is addressed by introducing the *long-term key reveal* query to the owner of the target session, before the session is completed. Earlier models such as the BR models and the CK model do not allow the adversary to reveal the long-term secret key of the owner of the target session before it is expired, and hence do not address this property.

- **Partial weak forward secrecy:** $\lambda -$ CAFL-freshness allows the adversary to corrupt either of the protocol principals, after the test session is activated. Hence, the CAFL model addresses partial weak forward secrecy. The eCK model and the Moriyama-Okamoto model allow the adversary to reveal the long-term secret keys of both protocol principals of the target session after the target session is activated, as long as the adversary is passive. Hence they address weak forward secrecy. The CK model allows the adversary to reveal the long-term secret keys of both protocol principals of the target session, after the session is expired but regardless of whether the adversary is passive or active. Therefore, the CK model address perfect forward secrecy.

4.3 Constructing CAFL-secure Key Exchange Protocols

We investigate how to construct CAFL-secure key exchange protocols. Table 4.1 shows the generic construction of protocol $\pi 1$, which is CAFL-secure. The protocol $\pi 1$ is a key agreement protocol, in which each of the principals randomly chooses its ephemeral secret key, encrypts it with the public key of the intended partner principal, and sends the encrypted message to the intended partner principal. After exchanging the ephemeral secrets both principals compute the session key with ephemeral secrets, identities of the two principals and the protocol message sequence, using a pseudo random function. Updating the secret keys of protocol principals is an essential ingredient in achieving CAFL security. For this generic protocol construction, the underlying public-key encryption scheme is chosen to be a continuous leakage-resilient public-key encryption scheme, which updates the secret key after each decryption operation. We use this public-key encryption scheme to achieve the continuous leakage resiliency of the key exchange protocol.

4.3.1 Protocol Construction

KG, Enc and Dec are the key generation, encryption and decryption algorithms of the underlying adaptively chosen ciphertext after-the-fact leakage (CCLA2) secure public-key cryptosystem PKE (Section 2.4.2). PRF is a pseudo random function (Section 2.3.5) which generates the session key of length k.

A (Initiator)		B (Responder)
	Initial Setup	
$sk_A, pk_A \leftarrow \mathrm{KG}(1^k)$		$sk_B, pk_B \leftarrow \mathrm{KG}(1^k)$
	Protocol Execution	
$r_A \leftarrow \{0,1\}^k$		$r_B \leftarrow \{0,1\}^k$
$C_A \leftarrow \mathrm{Enc}(pk_B, r_A)$	$\xrightarrow{A,C_A}$	$(sk'_B, r_A) \leftarrow \mathrm{Dec}(sk_B, C_A)$ $sk_B \leftarrow sk'_B$
$(sk'_A, r_B) \leftarrow \mathrm{Dec}(sk_A, C_B)$ $sk_A \leftarrow sk'_A$	$\xleftarrow{B,C_B}$	$C_B \leftarrow \mathrm{Enc}(pk_A, r_B)$
$K \leftarrow \mathrm{PRF}(r_A, A\|\|C_A\|\|B\|\|C_B)$ $\oplus \mathrm{PRF}(r_B, A\|\|C_A\|\|B\|\|C_B)$		$K \leftarrow \mathrm{PRF}(r_A, A\|\|C_A\|\|B\|\|C_B)$ $\oplus \mathrm{PRF}(r_B, A\|\|C_A\|\|B\|\|C_B)$
	K is the session key	

Table 4.1: Generic CAFL-secure protocol construction: Protocol $\pi 1$

The generic protocol construction presented in our initial publication of this

work [ABS14], is vulnerable against active adversary. Following we explain the reason: In that protocol construction, the inputs to the key derivation function KDF contain the two ephemeral values, r_A and r_B, and the identities of the initiator and the responder, A and B, respectively: $\text{KDF}(r_A||r_B, \perp, k, A||B)$. Assume that the target session is in A, if the adversary corrupts A, decrypts the protocol message from C_B, then re-encrypt the corresponding plaintext again using pk_A, that makes a different plaintext C'_B, due to probabilistic encryption. Then if the adversary sends C'_B to A, instead of C_B, and executes the rest of the protocol, it results that A's session and B's session are not matching, because the message C_B computed by B is different from the message C'_B received by A, but compute the same session key. Thus, the adversary can issue `SessionKeyReveal` query to the session at B and thus trivially learn the session key of the target session. Cremers [Cre11] showed that such attacks can be avoided by using the session identifier in the key derivation step together with other shared secrets. In this thesis we re-design the protocol accordingly to ensure that mismatching sessions do not compute same session keys. Thus, the session key is derived using a pseudo random function (two calls to the PRF) as $K \leftarrow \text{PRF}(r_A, A||C_A||B||C_B) \oplus \text{PRF}(r_B, A||C_A||B||C_B)$, such that it contains the session identifier $A||C_A||B||C_B$ as an input to the pseudo random function.

Here we use a multiple-call pseudo random function PRF, instead of a key derivation function KDF, to ensure that the adversary chosen r_A xor r_B, can be used in the session key derivation, when the presence of an active adversary. Since the input $\sigma = r_A||r_B$ to the KDF should be uniformly random in the security definition of the KDF, using adversary chosen r_A xor r_B in the σ of KDF input is not allowed.

4.3.2 Security Proof of the Protocol $\pi 1$ in the CAFL Model

Theorem 4.3.1. The protocol $\pi 1$ is $\lambda -$ CAFL-secure, whenever the underlying public-key cryptosystem PKE is CCLA2-secure and PRF is a pseudo random function.

Let $\mathcal{U} = \{U_1, \ldots, U_{N_P}\}$ be a set of N_P parties. Each party U_i owns at most N_s number of protocol sessions. Let $\mathcal{A}$ be any PPT adversary against the key exchange protocol $\pi 1$. Then the advantage of $\mathcal{A}$ against the CAFL-security of

the protocol $\pi 1$, $\mathrm{Adv}_{\pi 1}^{\lambda-\mathrm{CAFL}}$ is:

$$\mathrm{Adv}_{\pi 1}^{\lambda-\mathrm{CAFL}}(\mathcal{A}) \leq N_P^2 N_s^2 \big(\mathrm{Adv}_{\mathrm{PKE}}^{\mathrm{CCLA2}}(\mathcal{D}) + \mathrm{Adv}_{\mathrm{PRF}}(\mathcal{B})\big).$$

where $\mathcal{B}, \mathcal{D}$ are efficient algorithms constructed using the adversary $\mathcal{A}$, against the underlying pseudo random function, PRF, and the public-key cryptosystem, PKE, respectively.

In order to formally prove the CAFL-security of the protocol $\pi 1$ we use the game hopping technique [BR06, KR01, Sho04]; define a sequence of games and relate the adversary's advantage of distinguishing each game from the previous game to the advantage of breaking one of the underlying cryptographic primitive. The proof structure is similar to Boyd et al. [BCNP09].

Proof. Assume that the adversary $\mathcal{A}$ can win the challenge against the protocol $\pi 1$ challenger with advantage $\mathrm{Adv}_{\pi 1}^{\lambda-\mathrm{CAFL}}(\mathcal{A})$. We split the proof into two cases: partner to the test session exists and partner to the test session does not exist.

Case 1: Partner to the test session exists

In this case we consider three sub cases as follows:

1. Adversary corrupts the owner of the test session, but does not corrupt the peer.
2. Adversary corrupts the peer of the test session, but does not corrupt the owner.
3. Adversary corrupts neither the owner nor the partner of the test session

Case 1.1: Adversary corrupts the owner of the test session, but does not corrupt the peer

In this case we consider the situation that $\mathcal{A}$ corrupts the owner of the test session but not the partner.

Game 1. This game is the original game. When the `Test` query is asked, the Game 1 challenger chooses a random bit $b \xleftarrow{\$} \{0, 1\}$. If $b = 1$, the real session key is given to $\mathcal{A}$, otherwise a random value chosen from the same session key space is given.

Game 2. Same as Game 1 with the following exception: before $\mathcal{A}$ begins, two distinct random principals $U^*, V^* \leftarrow \{U_1, \ldots, U_{N_P}\}$ are chosen and two random numbers $s^*, t^* \leftarrow \{1, \ldots, N_s\}$ are chosen, where N_P is the number of protocol principals and N_s is the number of sessions on a principal. The session $\Pi^{s^*}_{U^*,V^*}$ is chosen as the *target session* and the session $\Pi^{t^*}_{V^*,U^*}$ is chosen as the *partner* to the target session. If the test session is *not* the session $\Pi^{s^*}_{U^*,V^*}$ or the partner to the session is not $\Pi^{t^*}_{V^*,U^*}$, the Game 2 challenger aborts the game.

Game 3. Same as Game 2 with the following exception: the Game 3 challenger chooses a random value $r' \xleftarrow{\$} \{0,1\}^k$.

- If the test session is on the initiator, the challenger computes the session key in the test session $K \leftarrow \mathrm{PRF}(r', U^*||C_{U^*}||V^*||C_{V^*}) \oplus \mathrm{PRF}(r_{V^*}, U^*||C_{U^*}||V^*||C_{V^*})$.
- If the test session is on the responder, the challenger computes the session key in the test session $K \leftarrow \mathrm{PRF}(r_{V^*}, V^*||C_{V^*}||U^*||C_{U^*}) \oplus \mathrm{PRF}(r', V^*||C_{V^*}||U^*||C_{U^*})$.

The session key is computed in the same way in the partner to the test session.

Game 4. Same as Game 3 with the following exception: In the `Test` query, in the target session, the Game 4 challenger randomly chooses $K \xleftarrow{\$} \{0,1\}^k$ and sends it to the adversary $\mathcal{A}$ as the answer to the `Test` query. In sessions at V^* which has the same incoming message to V^* as in the target session, the session key is randomly chosen as $K \xleftarrow{\$} \{0,1\}^k$.

Differences between games. In this section the adversary's advantage of distinguishing each game from the previous game is investigated. $\mathrm{Adv}_{\text{Game x}}(\mathcal{A})$ denotes the advantage of the adversary $\mathcal{A}$ of winning Game x.

Game 1 is the original game. Hence,

$$\mathrm{Adv}_{\text{Game 1}}(\mathcal{A}) = \mathrm{Adv}^{\lambda-\mathrm{CAFL}}_{\pi 1,\text{Case 1.1}}(\mathcal{A}). \tag{4.1}$$

Game 1 and Game 2. The probability of Game 2 to be halted due to incorrect choice of the test session is $1 - \frac{1}{N_P^2 N_s^2}$. Unless the incorrect choice happens, Game 2 is identical to Game 1. Hence,

$$\mathrm{Adv}_{\text{Game 2}}(\mathcal{A}) = \frac{1}{{N_P}^2 N_s^2}\mathrm{Adv}_{\text{Game 1}}(\mathcal{A}). \tag{4.2}$$

Game 2 and Game 3. We introduce an algorithm $\mathcal{D}$ which is constructed using the adversary $\mathcal{A}$. If $\mathcal{A}$ can distinguish the difference between Game 2 and Game 3, then $\mathcal{D}$ can be used against the CCLA2 challenger of the underlying public-key cryptosystem, PKE. The algorithm $\mathcal{D}$ uses the public key of the CCLA2 challenger as the public key of the protocol principal V^* and generates public/secret key pairs for all other protocol principals. $\mathcal{D}$ runs a copy of $\mathcal{A}$ and interacts with $\mathcal{A}$, such that $\mathcal{A}$ is interacting with either Game 2 or Game 3. $\mathcal{D}$ picks two random strings, $r'_0, r'_1 \leftarrow \{0,1\}^k$ and passes them to the CCLA2 challenger. From the CCLA2 challenger, $\mathcal{D}$ receives a challenge ciphertext C such that $C \leftarrow \mathrm{Enc}(pk_{V^*}, r')$ where $r' = r'_0$ or $r' = r'_1$. The following describes $\mathcal{D}$'s procedure of answering queries.

- `Send`(U, V, s, m, f) query:
  - $U = U^*, V = V^*, s = s^*$:
    - If U^* is the initiator, $\mathcal{D}$ sends the ciphertext C to $\mathcal{A}$ as the first message of the test session. Upon receiving the second protocol message computes the session key $K \leftarrow \mathrm{PRF}(r'_1, U^*||C_{U^*}||V^*||C_{V^*}) \oplus \mathrm{PRF}(r_{V^*}, U^*||C_{U^*}||V^*||C_{V^*})$.
    - If U^* is the responder, upon receiving the first protocol message sends C to $\mathcal{A}$, and computes the session key $K \leftarrow \mathrm{PRF}(r'_1, V^*||C_{V^*}||U^*||C_{U^*}) \oplus \mathrm{PRF}(r_{V^*}, V^*||C_{V^*}||U^*||C_{U^*})$.
  - $U = U^*, V = V^*, s \neq s^*$: Executes the protocol normally.
  - $U = U^*, V \neq V^*$: Executes the protocol normally.
  - $U = V^*$:
    - If this is the initiator and it is the first message, then executes the protocol normally.
    - If this is the initiator and the second protocol message, or the responder:
      - If C has come as the incoming message uses r'_1 as the decryption of the incoming message. To obtain the corresponding leakage, $\mathcal{D}$ first encrypts r'_1 using pk_{V^*}, gets that ciphertext

and access the leakage oracle of CCLA2 challenger with the ciphertext of r'_1.

· Else uses the decryption oracle to decrypt incoming messages.

– $U, V \neq U^*$ or V^*: Executes the protocol normally.

For all other leakage queries $f(sk_{V^*})$, $\mathcal{D}$ obtains the leakage accessing the leakage oracle of the CCLA2 challenger, whereas for all the other leakage queries, $\mathcal{D}$ can compute the leakage by its own, because $\mathcal{D}$ knows all other secret keys.

- `SessionKeyReveal`(U, V, s) query: `SessionKeyReveal` query is not allowed to the target session or the partner of the target session. $\mathcal{D}$ can compute all the session keys by executing the protocol.

  – For sessions involving the principal V^*, and the incoming message to V^* is the same message which has come to V^* in the target session, uses r'_1 as the decryption.

  – For other sessions involving the principal V^*, $\mathcal{D}$ can decrypt the incoming messages to V^* by using the decryption oracle.

  – Otherwise, $\mathcal{D}$ can decrypt all the other incoming messages to protocol principals by its own.

  Then compute the session key using the PRF.

- `EphemeralKeyReveal`(U, V, s) query: For all `EphemeralKeyReveal` queries allowed in the freshness condition, $\mathcal{D}$ can answer correctly, because $\mathcal{D}$ has the ephemeral keys.

- `Corrupt`(U) query: Except for V^*, algorithm $\mathcal{D}$ can answer all other `Corrupt` queries. In this case we consider the situation in which the adversary is not allowed to corrupt the partner principal of the target session, so in fact, $\mathcal{D}$ can answer all legitimate `Corrupt` queries.

- `Test`(U, V, s) query: Answers with the K which is computed at the `Send` query when $U = U^*, V = V^*, s = s^*$.

If r'_1 is the decryption of C in the target session, the simulation constructed by $\mathcal{D}$ is identical to Game 2 whereas if r'_0 is the decryption of C, the simulation

constructed by $\mathcal{D}$ is identical to Game 3. If $\mathcal{A}$ can distinguish the difference between Game 2 and Game 3, then $\mathcal{D}$ can distinguish whether $C \leftarrow \text{Enc}(pk_{V^*}, r'_0)$ or $C \leftarrow \text{Enc}(pk_{V^*}, r'_1)$.

The algorithm $\mathcal{D}$ plays the CCLA2 game against the public-key cryptosystem PKE according to the Definition 2.4.6 since $\mathcal{D}$ does not ask the decryption of the challenge ciphertext C. Hence,

$$|\text{Adv}_{\text{Game 2}}(\mathcal{A}) - \text{Adv}_{\text{Game 3}}(\mathcal{A})| \leq \text{Adv}^{\text{CCLA2}}_{\text{PKE}}(\mathcal{D}). \tag{4.3}$$

Game 3 and Game 4. If $\mathcal{A}$ can distinguish the difference between Game 3 and Game 4, then $\mathcal{A}$ can be used as a subroutine of an algorithm $\mathcal{B}$, which is used to distinguish whether the session key value K is computed using the real PRF with a hidden key, or using a random function. The adversary $\mathcal{A}$ is given a K, such that it is computed using the PRF or randomly chosen from the session key space. The following describes $\mathcal{B}$'s procedure of answering queries.

- $\texttt{Send}(U, V, s, m, f)$ query:
  - $U = U^*, V = V^*, s = s^*$:
    - If U^* is the initiator, upon receiving the second protocol message computes the session key $K \leftarrow \text{Oracle}^{\text{PRF}}(U^*||C_{U^*}||V^*||C_{V^*}) \oplus \text{PRF}(r_{V^*}, U^*||C_{U^*}||V^*||C_{V^*})$.
    - If U^* is the responder, upon receiving the first protocol message computes the session key $K \leftarrow \text{Oracle}^{\text{PRF}}(U^*||C_{U^*}||V^*||C_{V^*}) \oplus \text{PRF}(r_{V^*}, U^*||C_{U^*}||V^*||C_{V^*})$.
  - $U = U^*, V = V^*, s \neq s^*$: Executes the protocol normally.
  - $U = U^*, V \neq V^*$: Executes the protocol normally.
  - $U = V^*$:
    - If this is the initiator and it is the first message, then executes the protocol normally.
    - If this is the initiator and the second protocol message, or the responder:
      - If the same message that came to V^* in the test session has come as the incoming message, computes the session key using the $\text{Oracle}^{\text{PRF}}$.

· Otherwise, executes the protocol normally.

– $U, V \neq U^*$ or V^*: Executes the protocol normally.

For all leakage queries, $\mathcal{B}$ can compute the leakage by its own, because $\mathcal{B}$ knows all the secret keys.

- `SessionKeyReveal`(U, V, s) query: `SessionKeyReveal` query is not allowed to the target session or its partner. $\mathcal{B}$ can compute all the session keys by executing the protocol.

  – For sessions involving the principal V^*, and the incoming message to V^* is the same message which has come to V^* in the target session, $\mathcal{B}$ uses $\text{Oracle}^{\text{PRF}}$ to compute the session key.

  – For all other sessions, $\mathcal{B}$ computes the session key by using the PRF.

- `EphemeralKeyReveal`(U, V, s) query: $\mathcal{B}$ can answer all `EphemeralKeyReveal` queries, which are allowed by the freshness condition, because $\mathcal{B}$ has the ephemeral keys.

- `Corrupt`(U) query: Except for V^*, algorithm $\mathcal{B}$ can answer all other `Corrupt` queries. In this case we consider the situation in which the adversary is not allowed to corrupt the partner principal of the target session, so in fact, $\mathcal{B}$ can answer all legitimate `Corrupt` queries.

- `Test`(U, V, s) query: Answers with the K which is computed at the `Send` query when $U = U^*, V = V^*, s = s^*$.

If the oracle is using the real PRF with a hidden key, the simulation is identical to Game 3, whereas if the oracle is using a random function, the simulation constructed is identical to Game 4. If $\mathcal{A}$ can distinguish the difference between Game 3 and Game 4, then $\mathcal{A}$ can be used as a subroutine of an algorithm $\mathcal{B}$, which is used to distinguish whether the PRF challenger is real or random. Hence,

$$|\text{Adv}_{\text{Game 3}}(\mathcal{A}) - \text{Adv}_{\text{Game 4}}(\mathcal{A})| \leq \text{Adv}_{\text{PRF}}(\mathcal{B}). \tag{4.4}$$

Semantic security of the session key in Game 4. Since the session key K of $\Pi^{s^*}_{U^*,V^*}$ is chosen randomly and independently from all other values, $\mathcal{A}$ does not have any advantage in Game 4. Hence,

$$\mathrm{Adv}_{\text{Game 4}}(\mathcal{A}) = 0. \tag{4.5}$$

Combining the results above, we find,

$$\mathrm{Adv}^{\lambda-\mathrm{CAFL}}_{\pi 1,\text{Case 1.1}}(\mathcal{A}) \leq N_P^2 N_s^2 \big(\mathrm{Adv}^{\mathrm{CCLA2}}_{\mathrm{PKE}}(\mathcal{D}) + \mathrm{Adv}_{\mathrm{PRF}}(\mathcal{B})\big). \tag{4.6}$$

Case 1.2: Adversary corrupts the peer of the test session, but does not corrupt the owner.

In this case we consider the situation that $\mathcal{A}$ corrupts the partner of the test session but not the owner. The proof structure and games are similar to the previous case. The differences in this case is that the algorithm $\mathcal{D}$ uses the public key of the CCLA2 challenger as the public key of the protocol principal U^* (difference between Game 2 and Game 3), and $\mathrm{Oracle}^{\mathrm{PRF}}$ is used when the incoming message to U^* in the test session is used as the incoming message to U^* in any other sessions (Game 3 and Game 4 analysis). We find,

$$\mathrm{Adv}^{\lambda-\mathrm{CAFL}}_{\pi 1,\text{Case 1.2}}(\mathcal{A}) \leq N_P^2 N_s^2 \big(\mathrm{Adv}^{\mathrm{CCLA2}}_{\mathrm{PKE}}(\mathcal{D}) + \mathrm{Adv}_{\mathrm{PRF}}(\mathcal{B})\big). \tag{4.7}$$

Case 1.3: Adversary corrupts neither the owner nor the partner of the test session

In this case we consider the situation that $\mathcal{A}$ corrupts neither the owner nor the partner of the test session. So $\mathcal{D}$ can set the public key of the CCLA2 challenger as the public key of either U^* or V^*. The proof structure and games are similar to the previous case. We consider two sub cases under this case as follows:

- (a) Adversary does not ask `EphemeralKeyReveal`(V^*, U^*, t^*): simulation and analysis of this case is similar to the Case 1.2, because here $\mathcal{D}$ can set the public key of the CCLA2 challenger as the public key of the protocol principal U^* and proceed with the simulation as in the Case 1.2 (only difference is that here the adversary does not corrupt the partner principal of the test session, as in the Case 1.2, but rest of the simulation is same as

in the Case 1.2). Thus,

$$\mathrm{Adv}^{\lambda-\mathrm{CAFL}}_{\pi 1,\text{Case 1.3.a}}(\mathcal{A}) \leq N_P^2 N_s^2 \big(\mathrm{Adv}^{\mathrm{CCLA2}}_{\mathrm{PKE}}(\mathcal{D}) + \mathrm{Adv}_{\mathrm{PRF}}(\mathcal{B})\big). \tag{4.8}$$

- (b) Adversary does not ask `EphemeralKeyReveal`(U^*, V^*, s^*): simulation and analysis of this case is similar to the Case 1.1, because here $\mathcal{D}$ can set the public key of the CCLA2 challenger as the public key of the protocol principal V^* and proceed with the simulation as in the Case 1.1 (only difference is that here the adversary does not corrupt the owner of the test session, as in the Case 1.1, but rest of the simulation is same as in the Case 1.1). Thus,

$$\mathrm{Adv}^{\lambda-\mathrm{CAFL}}_{\pi 1,\text{Case 1.3.b}}(\mathcal{A}) \leq N_P^2 N_s^2 \big(\mathrm{Adv}^{\mathrm{CCLA2}}_{\mathrm{PKE}}(\mathcal{D}) + \mathrm{Adv}_{\mathrm{PRF}}(\mathcal{B})\big). \tag{4.9}$$

Case 2: Partner to the test session does not exist

Game 1. This game is the original game. When the `Test` query is asked, the Game 1 challenger chooses a random bit $b \leftarrow \{0,1\}$. If $b = 1$, the real session key is given to $\mathcal{A}$, otherwise a random value chosen from the same session key space is given.

Game 2. Same as Game 1 with the following exception: before $\mathcal{A}$ begins, two distinct random principals $U^*, V^* \leftarrow \{U_1, \ldots, U_{N_P}\}$ are chosen and a random number $s^* \leftarrow \{1, \ldots, N_s\}$ is chosen, where N_P is the number of protocol principals and N_s is the number of sessions on a principal. The session $\Pi^{s^*}_{U^*,V^*}$ is chosen as the *target session*. If the test session is *not* the session $\Pi^{s^*}_{U^*,V^*}$, the Game 2 challenger aborts the game.

Game 3. Same as Game 2 with the following exception: the Game 3 challenger chooses a random value $r' \xleftarrow{\$} \{0,1\}^k$.

- If the test session is on the initiator, the challenger computes the session key in the test session $K \leftarrow \mathrm{PRF}(r', U^*||C_{U^*}||V^*||C_{V^*}) \oplus \mathrm{PRF}(r_{V^*}, U^*||C_{U^*}||V^*||C_{V^*})$.
- If the test session is on the responder, the challenger computes the session key in the test session $K \leftarrow \mathrm{PRF}(r_{V^*}, V^*||C_{V^*}||U^*||C_{U^*}) \oplus \mathrm{PRF}(r', V^*||C_{V^*}||U^*||C_{U^*})$.

The session key is computed in the same way in the partner to the test session.

Game 4. Same as Game 3 with the following exception: In the `Test` query, in the target session the Game 4 challenger randomly chooses $K \xleftarrow{\$} \{0,1\}^k$ and sends it to the adversary $\mathcal{A}$ as the answer to the `Test` query. In sessions at V^* which has a same incoming message to V^* as in the target session, the session key is randomly chosen as $K \xleftarrow{\$} \{0,1\}^k$.

Differences between games. Game 1 is the original game. Hence,

$$\mathrm{Adv}_{\text{Game 1}}(\mathcal{A}) = \mathrm{Adv}^{\lambda-\mathrm{CAFL}}_{\pi 1,\text{Case 2}}(\mathcal{A}). \tag{4.10}$$

Game 1 and Game 2. The probability of Game 2 to be halted due to incorrect choice of the test session is $1 - \frac{1}{{N_P}^2 N_s}$. Unless the incorrect choice happens, Game 2 is identical to Game 1. Hence,

$$\mathrm{Adv}_{\text{Game 2}}(\mathcal{A}) = \frac{1}{{N_P}^2 N_s}\mathrm{Adv}_{\text{Game 1}}(\mathcal{A}). \tag{4.11}$$

Game 2 and Game 3. We introduce an algorithm $\mathcal{D}$ which is constructed using the adversary $\mathcal{A}$. If $\mathcal{A}$ can distinguish the difference between Game 2 and Game 3, then $\mathcal{D}$ can be used against the CCLA2 challenger of underlying public-key cryptosystem, PKE. The algorithm $\mathcal{D}$ uses the public key of the CCLA2 challenger as the public key of the protocol principal V^* and generates public/secret key pairs for all other protocol principals. $\mathcal{D}$ runs a copy of $\mathcal{A}$ and interacts with $\mathcal{A}$, such that it is interacting with either Game 2 or Game 3. $\mathcal{D}$ picks two random strings, $r'_0, r'_1 \leftarrow \{0,1\}^k$ and passes them to the CCLA2 challenger. From the CCLA2 challenger, $\mathcal{D}$ receives a challenge ciphertext C such that $C \leftarrow \mathrm{Enc}(pk_{V^*}, r')$ where $r' = r'_0$ or $r' = r'_1$. The following describes the procedure of answering queries.

- `Send`(U, V, s, m, f) query:
  - $U = U^*$, $V = V^*$ and $s = s^*$:
    - If U^* is the initiator, $\mathcal{D}$ sends the ciphertext C to $\mathcal{A}$ as the first message of the test session. Upon receiving the second protocol message computes the session key $K \leftarrow \mathrm{PRF}(r'_1, U^*||C_{U^*}||V^*||C_{V^*}) \oplus \mathrm{PRF}(r_{V^*}, U^*||C_{U^*}||V^*||C_{V^*})$.

* If U^* is the responder, upon receiving the first protocol message sends C to $\mathcal{A}$, and computes the session key $K \leftarrow \mathrm{PRF}(r'_1, V^* ||C_{V^*}||U^*||C_{U^*}) \oplus \mathrm{PRF}(r_{V^*}, V^*||C_{V^*}||U^*||C_{U^*})$.

– $U = U^*, V = V^*, s \neq s^*$: Executes protocol normally.

– $U = U^*, V \neq V^*$: Executes the protocol normally.

– $U = V^*$:

* If this is the initiator and it is the first message, then executes the protocol normally.

* If this is the initiator and the second protocol message, or the responder:

· If C has come as the incoming message uses r'_1 as the decryption of the incoming message. To obtain the corresponding leakage, $\mathcal{D}$ first encrypts r'_1 using pk_{V^*}, gets that ciphertext and access the leakage oracle of CCLA2 challenger with the ciphertext of r'_1.

· Else uses the decryption oracle to decrypt incoming messages.

– $U, V \neq U^*$ or V^*: Executes the protocol normally.

For all other leakage queries $f(sk_{V^*})$, $\mathcal{D}$ obtains the leakage accessing the leakage oracle, whereas for all other leakage $\mathcal{D}$ can compute the leakage by its own because $\mathcal{D}$ knows all other secret keys.

- `SessionKeyReveal`(U, V, s) query: `SessionKeyReveal` query is not allowed to the target session or its partner.

– For sessions involving the principal V^*, and the incoming message to V^* is the same message which has come to V^* in the target session, uses r'_1 as the decryption.

– For other sessions involving the principal V^*, $\mathcal{D}$ can decrypt the incoming messages to V^* by using the decryption oracle.

– Otherwise, $\mathcal{D}$ can decrypt all the other incoming messages to protocol principals by its own.

Then compute the session key using the PRF.

- `EphemeralKeyReveal`(U, V, s) query: For all `EphemeralKeyReveal` queries allowed in the freshness condition, $\mathcal{D}$ can answer correctly, because $\mathcal{D}$ has the ephemeral keys.

- `Corrupt`(U) query: Except for V^*, algorithm $\mathcal{D}$ can answer all other `Corrupt` queries. In this case we consider the situation in which the adversary is not allowed to corrupt the partner principal of the target session, so in fact, $\mathcal{D}$ can answer all `Corrupt` queries.

- `Test`(U, V, s) query: To compute the answer to the `Test`(U^*, V^*, s^*) query, the algorithm $\mathcal{D}$ uses r_1' as the decryption of the ciphertext C and computes the session key using the r_1' value as the ephemeral key of the principal U^*.

If r_1' is the decryption of C coming to, V^* in the test session, the simulation constructed by $\mathcal{D}$ is identical to Game 2 whereas if r_0' is the decryption of C, the simulation constructed by $\mathcal{D}$ is identical to Game 3. If $\mathcal{A}$ can distinguish the difference between Game 2 and Game 3, then $\mathcal{D}$ can distinguish whether $C \leftarrow \text{Enc}(pk_{V^*}, r_0')$ or $C \leftarrow \text{Enc}(pk_{V^*}, r_1')$.

The algorithm $\mathcal{D}$ plays the CCLA2 game against the public-key cryptosystem PKE according to the Definition 2.4.6 since $\mathcal{D}$ does not ask the decryption of the challenge ciphertext C. Hence,

$$|\text{Adv}_{\text{Game 2}}(\mathcal{A}) - \text{Adv}_{\text{Game 3}}(\mathcal{A})| \leq \text{Adv}_{\text{PKE}}^{\text{CCLA2}}(\mathcal{D}). \tag{4.12}$$

Game 3 and Game 4. If $\mathcal{A}$ can distinguish the difference between Game 3 and Game 4, then $\mathcal{A}$ can be used as a subroutine of an algorithm $\mathcal{B}$, which is used to distinguish whether the session key value K is computed K is computed using the real PRF with a hidden key, or using a random function. The adversary $\mathcal{A}$ is given a K, such that it is computed using the PRF or randomly chosen from the session key space. The following describes $\mathcal{B}$'s procedure of answering queries.

- `Send`(U, V, s, m, f) query:

  - $U = U^*$, $V = V^*$ and $s = s^*$:

    * If U^* is the initiator, upon receiving the second protocol message computes the session key $K \leftarrow \text{Oracle}^{\text{PRF}}(U^*||C_{U^*}||V^*||C_{V^*}) \oplus \text{PRF}(r_{V^*}, U^*||C_{U^*}||V^*||C_{V^*})$.

    * If U^* is the responder, upon receiving the first protocol message computes the session key $K \leftarrow \text{Oracle}^{\text{PRF}}(U^*||C_{U^*}||V^*||C_{V^*}) \oplus \text{PRF}(r_{V^*}, U^*||C_{U^*}||V^*||C_{V^*})$.
  - $U = U^*, V = V^*, s \neq s^*$: Executes protocol normally.
  - $U = U^*, V \neq V^*$: Executes the protocol normally.
  - $U = V^*$:
    * If this is the initiator and it is the first message, then executes the protocol normally.
    * If this is the initiator and the second protocol message, or the responder:
      · If the same message that came to V^* in the test session has come as the incoming message, computes the session key using the $\text{Oracle}^{\text{PRF}}$.
      · Otherwise, executes the protocol normally.
  - $U, V \neq U^*$ or V^*: Executes the protocol normally.

  For all leakage queries, $\mathcal{B}$ can compute the leakage by its own, because $\mathcal{B}$ knows all the secret keys.

- `SessionKeyReveal`(U, V, s) query: `SessionKeyReveal` query is not allowed to the target session or its partner.
  - For sessions involving the principal V^*, and the incoming message to V^* is the same message which has come to V^* in the target session, $\mathcal{B}$ uses $\text{Oracle}^{\text{PRF}}$ to compute the session key.
  - For all other sessions, $\mathcal{B}$ computes the session key by using the PRF.
- `EphemeralKeyReveal`(U, V, s) query: $\mathcal{B}$ can answer all `EphemeralKeyReveal` queries $\mathcal{B}$ which are allowed in the freshness condition, because $\mathcal{B}$ has the ephemeral keys.
- `Corrupt`(U) query: Except for V^*, algorithm $\mathcal{B}$ can answer all other `Corrupt` queries. In this case we consider the situation in which the adversary is not allowed to corrupt the partner principal of the target session, so in fact, $\mathcal{B}$ can answer all `Corrupt` queries.

- `Test`(U, V, s) query: Answers with the K computed in `Send` query when $U = U^*$, $V = V^*$ and $s = s^*$.

If the oracle is using the real PRF with a hidden key, the simulation is identical to Game 3, whereas if the oracle is using a random function, the simulation constructed is identical to Game 4. If $\mathcal{A}$ can distinguish the difference between Game 3 and Game 4, then $\mathcal{A}$ can be used as a subroutine of an algorithm $\mathcal{B}$, which is used to distinguish whether the PRF challenger is real or random. Hence,

$$|\mathrm{Adv}_{\text{Game }3}(\mathcal{A}) - \mathrm{Adv}_{\text{Game }4}(\mathcal{A})| \leq \mathrm{Adv}_{\mathrm{PRF}}(\mathcal{B}). \tag{4.13}$$

Semantic security of the session key in Game 4. Since the session key K of $\Pi^{s^*}_{U^*,V^*}$ is chosen randomly and independently from all other values, $\mathcal{A}$ does not have any advantage in Game 4. Hence,

$$\mathrm{Adv}_{\text{Game }4}(\mathcal{A}) = 0. \tag{4.14}$$

Combining the results above, we find,

$$\mathrm{Adv}^{\lambda-\mathrm{CAFL}}_{\pi 1,\text{Case }2}(\mathcal{A}) \leq N_P^2 N_s \big(\mathrm{Adv}^{\mathrm{CCLA2}}_{\mathrm{PKE}}(\mathcal{D}) + \mathrm{Adv}_{\mathrm{PRF}}(\mathcal{B})\big). \tag{4.15}$$

Combine Case 1 and Case 2

According to the analysis we can see the adversary $\mathcal{A}$'s advantage of winning against the protocol $\pi 1$ challenger is

$$\mathrm{Adv}^{\lambda-\mathrm{CAFL}}_{\pi 1}(\mathcal{A}) \leq N_P^2 N_s^2 \big(\mathrm{Adv}^{\mathrm{CCLA2}}_{\mathrm{PKE}}(\mathcal{D}) + \mathrm{Adv}_{\mathrm{PRF}}(\mathcal{B})\big).$$

□

4.3.3 Leakage Tolerance of the CAFL-secure Protocol $\pi 1$

In the created protocol, a principal simply encrypts a randomly-chosen ephemeral key using a CCLA2-secure public key encryption scheme, and sends it to the partner principal. Therefore, the leakage tolerance is exactly same as the leakage tolerance of the underlying CCLA2-secure public key encryption scheme.

Dziembowski and Faust. [DF11] constructed a CCLA2-secure public-key cryptosystem, where the secret key $sk = (x_1, x_2) \in (\mathbb{Z}_q^*)^2$ is split into two parts ℓ_{sk}, r_{sk}

such that $\ell_{sk} \xleftarrow{\$} (\mathbb{Z}_q^*)^n$ at random and $r_{sk} \leftarrow (\mathbb{Z}_q^*)^{n\times 2}$ holding $\ell_{sk} \cdot r_{sk} = sk$, where n is the statistical security parameter. They proved their public-key cryptosystem is CCLA2-secure for $\lambda = 0.15 \cdot n \cdot \log q - 1$. So if we consider $n = 20$ and $\log(q-1)$ to be 1024, we can allow $\lambda = 3072$ bits of leakage, from each split per occurrence. Considering only the most expensive computations, the computation cost of Enc and Dec is 5**Exp** for each, where **Exp** is the computational cost of an exponentiation.

4.4 Summary

We have presented a security model and a key exchange protocol that improves the amount and type of secret leakage. Our security model allows the adversary to fully compromise a variety of long-term and short-term ephemeral values, as well as to obtain partial, adaptive, continuous, after-the-fact leakage of long-term secret keys. Previous key exchange security models either do not consider partial leakage at all (BR, CK, eCK) or allow only bounded leakage before the test session is queried and none after (Moriyama–Okamoto). Our model captures a wide variety of practical attack scenarios, including cold boot, key compromise impersonation, and side channel attacks. We have presented a generic key exchange protocol, that relies on a CCLA2-secure public-key encryption scheme, and can achieve the same amount of leakage tolerance as the underlying public-key encryption scheme.

In the next chapter we will strengthen the security model by introducing an eCK-like freshness condition and construct a leakage-resilient key exchange protocol which can be proven secure in that stronger security model.

Chapter 5

Bounded/Continuous After-the-fact Leakage eCK Model

Contents

In Chapter 4 we advanced modelling security by addressing more granular partial leakage of long-term secret keys even after the test session is activated, but enforcing additional restrictions to the freshness condition. In this chapter, we address partial leakage of long-term secrets of protocol principals, even after the session key is established, while stripping-off the additional restrictions to the freshness condition. We introduce a generic key exchange security model, which can be instantiated allowing bounded or continuous leakage, even when the adversary learns certain ephemeral secrets or session keys. Our model is the strongest known partial-leakage-based security model for key exchange protocols. We present a generic construction of a two-pass leakage-resilient key exchange protocol that is secure in slightly weakened version of the presented generic security model, by introducing a new concept: the leakage-resilient NAXOS trick. We identify a special property for public-key cryptosystems: *pair generation indistinguishability*, and show how to obtain the leakage-resilient NAXOS trick from a pair generation indistinguishable leakage-resilient public-key cryptosystem. The generic protocol can be instantiated with certain available public-key encryption schemes and signature schemes in the literature. Further, we present a concrete construction of a key exchange protocol, which can be

proven secure in the continuous leakage variant of the presented model. The concrete protocol is constructed using a leakage-resilient storage scheme and its refreshing protocol, and is based on the Diffie-Hellman key exchange.

5.1 Introduction

Previous key exchange models including leakage have been limited in one or more ways. In most of the models [ADW09, DHLAW10, MO11] the total amount of leakage allowed is bounded, which is troublesome because a protocol may reveal a limited amount of leakage each time it runs, and hence reveal "continuous" leakage. In addition, the adversary cannot obtain any leakage information after the session key is established for the session which the adversary targets for its attack. This is problematic because it does not address the security of one session, even if some leakage happens in subsequent sessions. A recent paper [YMSW13] uses a different leakage model with allows auxiliary input [DKL09] but this cannot be directly compared with other leakage models. Although this model allows the adversary to make leakage queries on the complete secret, the values returned to the adversary are limited to those which are hard to invert and therefore are of limited use to the adversary.

Moreover, the continuous after-the-fact leakage key exchange security model (CAFL) we presented in the Chapter 4 enforces more restrictions to the freshness definition, more than in the eCK model [LLM07] or Moriyama-Okamoto model [MO11]: it does not allow to reveal the ephemeral keys of both principals as to corrupt both protocol principals of the target session etc. So we need to accommodate a reasonable security model which addresses more granular leakage and at the same time does not enforce more restrictions than currently existing key exchange security models.

In this chapter, we construct a *generic leakage-security model* for key exchange protocols, which can be instantiated as a *bounded* leakage variant as well as a *continuous* leakage variant. In the bounded leakage variant, the total amount of leakage is bounded, whereas in the continuous leakage variant, a protocol execution may reveal a small amount of leakage each time. Further, the adversary is allowed to obtain the leakage even after the session key is established for the session in which the adversary tries to distinguish the real session key from a random session key. We emphasize that the leakage functions are arbitrary

polynomial time functions with output length restrictions. Thus, our approach allows *after-the-fact leakage* in bounded or continuous leakage model.

5.2 After-the-fact Leakage-eCK ((·)AFL-eCK) Model

The generic after-the-fact leakage eCK ((·)AFL-eCK) model can be instantiated in two different ways which leads to two security models. Namely, *bounded* after-the-fact leakage eCK (BAFL-eCK) model and *continuous* after-the-fact leakage eCK (CAFL-eCK) model. The BAFL-eCK model allows the adversary to obtain a bounded amount of leakage of the long-term secret keys of the protocol principals, as well as reveal session keys, long-term secret keys and ephemeral keys. Differently, the CAFL-eCK model allows the adversary to continuously obtain arbitrarily large amount of leakage of the long-term secret keys of the protocol principals, enforcing the restriction that the amount of leakage per observation is bounded.

In both instantiations of the generic (·)AFL-eCK model the partnering definition and the adversarial powers are same. The freshness conditions differ according to the leakage allowed. So we can define the partnering and adversarial powers in the generic (·)AFL-eCK model and define the freshness separately in each BAFL-eCK and CAFL-eCK models.

5.2.1 Modelling Leakage

Considering side-channel attacks which can be mounted against key exchange protocols, the most realistic way is to obtain the leakage information of secret keys from the protocol computations which use secret keys for computations. Following the premise "only computation leaks information", we have modeled the leakage in a place where a computation takes place on secret keys. After issuing a `Send` query, the adversary will get a protocol message which is computed according to the normal protocol computations. So sending an adversary-chosen adaptive leakage function with the `Send` query reflects the premise "only computation leaks information".

We introduce a tuple of $\tilde{n}$ adaptively chosen efficiently computable leakage functions $\mathbf{f} = (f_{1j}, f_{2j}, \ldots, f_{\tilde{n}j})$; the size $\tilde{n}$ of the tuple is *protocol-specific*, and j indicates the j^{th} leakage occurrence. A key exchange protocol may use more than one cryptographic primitive and each primitive uses a distinct secret key or secret

state (in signature schemes). Hence, we need to address the leakage of secret keys or secret states from each of those primitives. Also, some cryptographic primitives which have been used to construct a key exchange protocol may be stateful cryptographic primitives. The execution of a stateful cryptographic primitive is split into a number of sequential stages and each of these stages uses one part of the secret key. The tuple of leakage functions $\mathbf{f} = (f_{1j}, f_{2j}, \ldots, f_{\tilde{n}j})$ leaks information from the secret key of each of the underlying primitives or each split of the secret keys at occurrence j. There exists a leakage parameter $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_{\tilde{n}})$ where each λ_i bounds the leakage for the corresponding primitive as key split.

5.2.2 Adversarial Powers

The adversary $\mathcal{A}$ is a probabilistic polynomial time (PPT) algorithm that controls the whole network. $\mathcal{A}$ interacts with a set of sessions which represent protocol instances. The following query allows the adversary $\mathcal{A}$ to run the protocol.

- `Send`$(U, V, s, m, \mathbf{f})$ query: The session $\Pi^s_{U,V}$, computes the next protocol message according to the protocol specification and sends it to the adversary $\mathcal{A}$, along with the leakage $\mathbf{f}(sk_U)$. $\mathcal{A}$ can also use this query to activate a new protocol instance as an initiator with blank m.

The following set of queries allow the adversary $\mathcal{A}$ to compromise certain session specific ephemeral secrets and long-term secrets from the protocol principals.

- `SessionKeyReveal`(U, V, s) query: $\mathcal{A}$ is given the session key of the session $\Pi^s_{U,V}$.
- `EphemeralKeyReveal`(U, V, s) query: $\mathcal{A}$ is given the ephemeral keys (per-session randomness) of the session $\Pi^s_{U,V}$.
- `Corrupt`(U) query: $\mathcal{A}$ is given the long-term secrets of the principal U. This query does not reveal any session keys or ephemeral keys to $\mathcal{A}$.

Once the session $\Pi^s_{U,V}$ has accepted a session key, the adversary $\mathcal{A}$ attempt to distinguish it from a random session key by asking the following query. The `Test` query is used to formalize the notion of the semantic security of a key exchange protocol.

- `Test`(U, s) query: When $\mathcal{A}$ asks the `Test` query, the challenger first chooses a random bit $b \xleftarrow{\$} \{0, 1\}$ and if $b = 1$ then the actual session key is returned to $\mathcal{A}$, otherwise a random string chosen from the same session key space is returned to $\mathcal{A}$. This query is only allowed to be asked once across all sessions.

Remark 5.2.1 (`Corrupt` query vs Leakage queries)**.** By issuing a `Corrupt` query, the adversary gets the party's entire long-term secret key. Separately, by issuing leakage queries (using a tuple leakage function $\mathbf{f}$ embedded with the `Send` query) the adversary gets respectively $\boldsymbol{\lambda}$-bounded leakage information about the long-term secret key(s). It may seem paradoxical to consider `Corrupt` and Leakage queries at the same time. But there is a good reason to consider both.

The eCK model addresses KCI attacks, because the adversary is allowed to corrupt the owner of the test session before the activation of the test session. In the generic $(\cdot)$AFL-eCK model, we allow the adversary to obtain leakage from the partner of the test session, in addition to allowing the adversary to corrupt the owner of the test session.

Hence, the generic $(\cdot)$AFL-eCK model allows the adversary to obtain more information than the eCK model. Moreover, none of the existing security models such as BR, CK, CK_{HMQV}, eCK allow a `Send` query with a tuple leakage function $\mathbf{f}$. Hence, we can see that $(\cdot)$AFL-eCK allows the adversary to obtain leakage information which none of the existing security models allow. Further, we emphasize that the technique of sending a tuple leakage function $\mathbf{f}$ with the `Send` query can be applied to any of the existing key exchange security models to obtain their leakage versions.

5.2.3 Bounded After-the-fact Leakage-eCK (BAFL-eCK) Model

In the BAFL-eCK model the total amount of leakage of each secret key of the underlying cryptographic primitives or each split of the secret key of the underlying stateful cryptographic primitives are bounded by leakage parameters. The leakage parameters are *primitive-specific*.

If the total leakage bound of the i^{th} cryptographic primitive (or the total leakage bound of the i^{th} state of the stateful cryptographic primitive) is λ_i and the leakage function f_{ij} outputs leakage bits of the secret key of the i^{th} cryptographic

primitive (or leakage bits of the i^{th} split of the secret key), then for leakage resilience of i^{th} cryptographic primitive (or the stateful cryptographic primitive), we need that $\sum_j |f_{ij}(s_i)| \leq \lambda_i$.

Definition 5.2.1 ($\boldsymbol{\lambda}-$ BAFL-eCK-freshness)**.** Let $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_{\tilde{n}})$ be a vector of $\tilde{n}$ elements (same size as $\mathbf{f}$ in `Send` query). A session $\Pi^{s}_{U,V}$ is said to be $\boldsymbol{\lambda}-$ BAFL-eCK-fresh if and only if:

1. The session $\Pi^{s}_{U,V}$ or its partner, $\Pi^{s'}_{V,U}$ (if it exists) has not been asked a `SessionKeyReveal`.
2. If the partner $\Pi^{s'}_{V,U}$ exists, none of the following combinations have been asked:
   (a) `Corrupt`(U) and `EphemeralKeyReveal`(U, V, s).
   (b) `Corrupt`(V) and `EphemeralKeyReveal`(V, U, s').
3. If the partner $\Pi^{s'}_{V,U}$ does not exist, none of the following combinations have been asked:
   (a) `Corrupt`(V).
   (b) `Corrupt`(U) and `EphemeralKeyReveal`(U, V, s).
4. For all `Send`$(U, \cdot, \cdot, \cdot, \mathbf{f})$ queries, $\sum_j |f_{ij}(sk_{U_i})| \leq \lambda_i$.
5. For all `Send`$(V, \cdot, \cdot, \cdot, \mathbf{f})$ queries, $\sum_j |f_{ij}(sk_{V_i})| \leq \lambda_i$.

5.2.4 Continuous After-the-fact Leakage-eCK (CAFL-eCK) Model

In the CAFL-eCK model, continuous leakage of each secret key of the underlying cryptographic primitives or each split of the secret key of the underlying stateful cryptographic primitives is allowed. The only restriction is that the amount of leakage per occurrence is bounded by leakage parameters. The leakage parameters are *primitive-specific*.

If the leakage bound of the i^{th} cryptographic primitive is λ_i per leakage occurrence and the leakage function f_{ij} outputs leakage bits of the secret key of the i^{th} cryptographic primitive, then for leakage resilience of i^{th} cryptographic

primitive we need that $|f_{ij}(sk_i)| \leq \lambda_i$, per leakage occurrence. If the leakage bound of the i^{th} state of the stateful cryptographic primitive is λ_i per leakage occurrence and the leakage function f_{ij} outputs leakage bits of the i^{th} split of the secret key, then for leakage resilience of the stateful cryptographic primitive we need that $|f_{ij}(sk_i)| \leq \lambda_i$, per leakage occurrence.

Definition 5.2.2 ($\boldsymbol{\lambda}-$ CAFL-eCK-freshness)**.** Let $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_{\tilde{n}})$ be a vector of $\tilde{n}$ elements (same size as $\mathbf{f}$ in `Send` query). A session $\Pi^s_{U,V}$ is said to be $\boldsymbol{\lambda}-$ CAFL-eCK-fresh if and only if: Conditions (1)-(3) of Definition 5.2.1 hold, and

4. For each $\texttt{Send}(U, \cdot, \cdot, \cdot, \mathbf{f})$ query, size of the output of $|f_{ij}(sk_{Ui})| \leq \lambda_i$.
5. For each $\texttt{Send}(V, \cdot, \cdot, \cdot, \mathbf{f})$ queries, size of the output of $|f_{ij}(sk_{Vi})| \leq \lambda_i$.

5.2.5 Defining Security

In this section we give formal definitions for partner sessions and security in the (·)AFL-eCK model.

Definition 5.2.3 (Partner sessions in generic (·)AFL-eCK model)**.** Two sessions $\Pi^s_{U,V}$ and $\Pi^{s'}_{U',V'}$ are said to be partners if all of the following hold:

1. both $\Pi^s_{U,V}$ and $\Pi^{s'}_{U',V'}$ have computed session keys;
2. messages sent from $\Pi^s_{U,V}$ and messages received by $\Pi^{s'}_{U',V'}$ are identical;
3. messages sent from $\Pi^{s'}_{U',V'}$ and messages received by $\Pi^s_{U,V}$ are identical;
4. $U' = V$ and $V' = U$;
5. Exactly one of U and V is the initiator and the other is the responder.

The protocol is said to be *correct* if two partner sessions compute and accept identical session keys.

We introduce the security game of the generic (·)AFL-eCK model. If we consider $\boldsymbol{\lambda}-$ BAFL-eCK-freshness, the security game is BAFL-eCK, otherwise if we consider $\boldsymbol{\lambda}-$ CAFL-eCK-freshness, it is CAFL-eCK security game.

Definition 5.2.4 ($\boldsymbol{\lambda}-$ (·)AFL-eCK security game)**.** Security of a key exchange protocol in the generic (·)AFL-eCK model is defined using the following security game, which is played by PPT adversary $\mathcal{A}$ against the protocol challenger.

- **Stage 0:** The challenger generates the keys by using the security parameter k.
- **Stage 1:** $\mathcal{A}$ may ask any of `Send`, `SessionKeyReveal`, `EphemeralKeyReveal` and `Corrupt` queries to any session at will.
- **Stage 2:** $\mathcal{A}$ chooses a $\boldsymbol{\lambda}-(\cdot)$AFL-eCK-fresh session and asks a `Test` query.
- **Stage 3:** $\mathcal{A}$ may continue asking `Send`, `SessionKeyReveal`, `Corrupt` and `EphemeralKeyReveal` queries. $\mathcal{A}$ may not ask a query that violates the $\boldsymbol{\lambda}-(\cdot)$AFL-eCK-freshness of the test session.
- **Stage 4:** At some point $\mathcal{A}$ outputs the bit $b' \in \{0,1\}$ which is its guess of the value b on the test session. $\mathcal{A}$ wins if $b' = b$.

$\text{Succ}_{\mathcal{A}}$ is the event that the adversary $\mathcal{A}$ wins the security game in Definition 5.2.4.

Definition 5.2.5 ($\boldsymbol{\lambda}-(\cdot)$AFL-eCK-security)**.** A protocol π is said to be $\boldsymbol{\lambda}-(\cdot)$AFL-eCK-secure if there is no PPT algorithm $\mathcal{A}$ that can win the $\boldsymbol{\lambda}-(\cdot)$AFL-eCK security game with non-negligible advantage. The advantage of an adversary $\mathcal{A}$ is defined as $\text{Adv}_{\pi}^{\boldsymbol{\lambda}-(\cdot)\text{AFL-eCK}}(\mathcal{A}) = |2\Pr(\text{Succ}_{\mathcal{A}}) - 1|$.

The generic (·)AFL-eCK model addresses the real world attack scenarios which were discussed in section 4.2.4, with the following differences.

- **Cold boot attacks:** The (·)AFL-eCK model allows the adversary to reveal either the long-term secret key (`Corrupt` query) or the ephemeral secret key (`EphemeralKeyReveal` query) of the target session. The bounded leakage instantiation of the (·)AFL-eCK model, BAFL-eCK model, allows bounded amount of leakage of the long-term secret key with the ephemeral key reveal. Thus these queries address the cold boot attacks to some extent, where the cold boot attacks reveal either (i) the long-term secret key, (ii) the ephemeral secret key, or (iii) the ephemeral secret key and part of the long-term secret key of a protocol principal, which is same as in the Moriyama-Okamoto model. The improvement of the BAFL-eCK model is that it allows the adversary to obtain the partial leakage of long-term secret key even after the test session is established, which is not allowed in the Moriyama-Okamoto model.

- **Weak forward secrecy:** (·)AFL-eCK-freshness allows the adversary to corrupt both of the protocol principals of the target session, after the test session is activated, as long as the adversary is passive. Hence, the (·)AFL-eCK model addresses weak forward secrecy, as for the eCK model and the Moriyama-Okamoto model.

- **eCK security:** The generic (·)AFL-eCK model is a leakage-resilient version of the eCK model [LLM07], hence, the generic (·)AFL-eCK model captures all possible attacks from ephemeral and long-term key compromises. More precisely, in sessions where the adversary does not modify the communication between parties (passive sessions), the adversary is allowed to reveal both ephemeral secrets, both long-term secrets, or one of each from two different parties, whereas in sessions where the adversary may forge the communication of one of the parties (active sessions), the adversary is allowed to reveal the long-term or ephemeral key of the other party.

The main reason to introduce a generic security model, (·)AFL-eCK model, and then present two instantiations (BAFL-eCK model and CAFL-eCK model) is to offer more flexibility to construct leakage-resilient key exchange protocols. The generic (·)AFL-eCK model gives a reasonable security framework for key exchange protocols capturing a wide range of practical attacks including side-channel attacks. The only difference between the two instantiations is the leakage allowance (bounded or continuous). If we need to implement a key exchange protocol which is resilient to cold boot attacks we use the BAFL-eCK model as the security framework, whereas if we need to implement a key exchange protocol which is secure against continuous-leakage side-channel attacks such as timing, power analysis and EM radiation, we use the CAFL-eCK model as the security framework.

5.3 Generic Construction of (·)AFL-eCK-secure Key Exchange Protocol

We investigate how to construct (·)AFL-eCK-secure key exchange protocols. The motivation of LaMacchia et al. [LLM07] in designing the eCK model was that an adversary should have to compromise both the long-term and ephemeral secret keys of a party in order to recover the session key. In their NAXOS protocol, the

main way this is accomplished is using what is now called the "NAXOS trick": a "pseudo" ephemeral key $\widetilde{esk}$ is computed as the hash of the long-term key lsk and the actual ephemeral key esk: $\widetilde{esk} \leftarrow H(esk, lsk)$. The value $\widetilde{esk}$ is never stored, and thus in the eCK model the adversary must learn both esk and lsk in order to be able to compute $\widetilde{esk}$. Note however, that in the NAXOS protocol, the initiator must compute $\widetilde{esk} = H(esk, lsk)$ twice: once when sending its Diffie–Hellman ephemeral public key $g^{\widetilde{esk}}$, and once when computing the Diffie–Hellman shared secrets from the received values. This is to avoid storing a single value that, when compromised, can be used to compute the session key.

5.3.1 Leakage-Resilient NAXOS Trick

Moving to the leakage-resilient setting requires rethinking the NAXOS trick. In the model "only computation leaks information", we must consider leakage at any place the long-term secret key is used. Thus, we need some kind of *leakage-resilient NAXOS trick*. As noted above, the initiator must not store the pseudo-ephemeral value, $\widetilde{esk}$, and instead must apply the NAXOS trick twice for each session. We replace the hash function H with a new leakage-resilient NAXOS trick to compute the pseudo-ephemeral value. The requirement is, given the long-term secret key and a particular ephemeral key, the NAXOS trick should always compute the same pseudo-ephemeral value, such that without knowing both the long-term and ephemeral keys the adversary is unable to compute the pseudo-ephemeral value. Moreover, the NAXOS trick computation should be resilient to the leakage of the long-term secret key, which happens even after the test session is activated.

A leakage-resilient NAXOS trick can be achieved by using the *decryption* function of a CPLA2-secure public-key cryptosystem [HL11]. Since decryption is deterministic, given the long-term secret key and a randomly chosen ciphertext, it will output the corresponding plaintext. So one can randomly choose an ephemeral key and use it as the ciphertext to the decryption function, and obtain the corresponding plaintext (output of the decryption function) as the pseudo-ephemeral value. Without knowing both the long-term and ephemeral keys, it is infeasible to compute the pesudo-ephemeral value. Thus, a leakage-resilient NAXOS trick can be achieved and the pseudo-ephemeral value can be computed. Further, *bounded* or *continuous* leakage-resilient key exchange protocol can be constructed, if the underlying public-key cryptosystem is bounded or continuous

leakage-resilient.

5.3.2 Pair Generation Indistinguishability

Using a decryption algorithm of a CPLA2-secure public-key cryptosystem does not work for our requirement unless the public-key cryptosystem has a special property: any randomly chosen ciphertext should be decrypted without rejection. A randomly chosen ciphertext can be rejected with a significant probability if NIZK proofs have been used for CPLA2-secure public-key cryptosystems. In NIZK proofs, the party which creates a ciphertext should provide a proof of knowledge of the plaintext, and the party which decrypts the ciphertext first verifies the proof, then only if the proof is correct it decrypts the ciphertext, otherwise rejects. Use of a CPLA2-secure public-key cryptosystem without the special property would allow the adversary to break the protocol with a significant probability, whenever a randomly chosen ciphertext is rejected. We formally define the special property as *pair generation indistinguishability*.

Definition 5.3.1 (Pair Generation Indistinguishability)**.** Let PKE = (KeyGen, Enc, Dec) be a public-key cryptosystem. For $(p, s) \xleftarrow{\$} \text{KeyGen}(1^k)$, let $D_1^{(p,s)}$, $D_2^{(p,s)}$ be two distributions such that $D_1^{(p,s)} = \{(m, c) : m \xleftarrow{\$} M, c \xleftarrow{\$} \text{Enc}(p, m)\}$ and $D_2^{(p,s)} = \{(m, c) : c \xleftarrow{\$} \text{C}, m \leftarrow \text{Dec}(s, c)\}$ where M is the message space and C is the ciphertext space. For $\epsilon \geq 0$, the public-key cryptosystem PKE is *ϵ-pair-generation-indistinguishable* ($\epsilon - \text{PG-IND}$) if for all $(p, s) \xleftarrow{\$} \text{KeyGen}(1^k)$, $\text{SD}(D_1^{(p,s)}, D_2^{(p,s)}) \leq \epsilon$.

Recall that the statistical distance, SD, between two distributions X and Y over a domain U is defined as $\text{SD}(X, Y) = \frac{1}{2} \sum_{u \in U} \big| \Pr[X = u] - \Pr[Y = u] \big|$.

The notion of pair generation indistinguishability shares some resemblance with the pseudorandom decapsulation notion introduced by Abdalla et al. [ACF09], where the notion was needed for the construction of verifiable random functions from identity-based key encapsulation schemes. They presented a methodology to construct verifiable random functions (VRFs) from a class of identity based key encapsulation mechanisms (IB-KEM) that is called VRF suitable. An IB-KEM is VRF suitable if it provides an unique decryption (i.e. given a ciphertext C produced with respect to an identity ID, all the secret keys corresponding to identity ID′, decrypt to the same value, even if ID = ID′) and it satisfies an additional property that is called *pseudorandom decapsulation*. Pseudorandom

decapsulation means that if one decrypts a ciphertext C, produced with respect to an identity ID, using the decryption key corresponding to any other identity ID$'$ the resulting value looks random to a polynomially bounded observer. Both the pair generation indistinguisbability and the pseudorandom decapsulation notions are similar, except that the pseudorandom decapsulation is in the identity-based setting whereas the pair generation indistinguishability is in the public key setting.

Now we show a $0-$PG-IND public-key cryptosystem available in the literature. Naor and Segev [NS09] described the framework of a hash proof system [CS02] as a key-encapsulation mechanism using the notion of Kiltz et al. [KPSY09]. Let K be the symmetric key space, C be the valid ciphertext space and M be the message space. Both K and C are the same size and elements of M are μ-bit strings. The leakage-resilient public-key cryptosystem of Naor and Segev encrypts an arbitrary message $m \xleftarrow{\$} \mathrm{M}$ as $(\Psi, c, seed)$, where $c \xleftarrow{\$} \mathrm{C}$ with the corresponding witness ω (of the fact that c is indeed a valid ciphertext from C), $seed \xleftarrow{\$} \{0,1\}^t$ is a random seed and $\Psi = Ext(Pub(p, c, \omega), seed) \oplus m$. $Ext : \mathrm{K} \times \{0,1\}^t \rightarrow \{0,1\}^\mu$ is a public average-case strong extractor function [DRS04], p is the public key and Pub is the deterministic public evaluation function of the underlying key-encapsulation mechanism. Pub receives as input a public key p, a valid ciphertext $c \in \mathrm{C}$ and the corresponding witness ω, and outputs an encapsulated key in K. Whenever a random $(\Psi, c, seed)$ is sampled, the decryption, $m \leftarrow \Psi \oplus Ext(Priv(s, c), seed)$ corresponds to a random $m \in \mathrm{M}$. $Priv$ is a private evaluation algorithm of the underlying key-encapsulation mechanism, receives as input the secret key s (of the public key p) and a valid ciphertext c, and outputs an encapsulated key in K. Thus, the leakage-resilient public-key cryptosystem of Naor and Segev is $0-$PG-IND. The generic CPLA2-secure public-key cryptosystem of Halevi and Lin [HL11] can be instantiated using the leakage-resilient public-key cryptosystem of Naor and Segev Hence, instantiation of the generic CPLA2-secure public-key cryptosystem of Halevi and Lin is also $0-$PG-IND.

5.3.3 Authenticating Protocol Messages

After computing the pseudo-ephemeral value by the NAXOS trick, a principal computes a Diffie-Hellman exponentiation and sends it to the other protocol principal. If that value is sent alone, the protocol is not secure because there is no authentication for the protocol messages, and hence an attacker can simply replace the original protocol message with its own value. In order to prevent this,

we need to provide authenticity to the protocol messages. There are unforgeable against chosen message leakage (UFCMLA) secure signature schemes available in the literature [KV09, FKPR09, MTVY11, BKKV10], which we can use to sign the protocol messages and provide authenticity. Further, the key exchange protocol is *bounded* or *continuous* leakage-resilient, if the underlying signature scheme is bounded or continuous leakage-resilient.

Weakening the $(\cdot)$AFL-eCK **Model**

When we consider the `EphemeralKeyReveal` query in our $(\cdot)$AFL-eCK model, it allows the adversary to learn the randomness used in the session, including the randomness used in signing. Full leakage of the randomness is not allowed in leakage-resilient signature schemes. Thus, in order to use available leakage-resilient signature schemes in our protocol instantiation, we will assume that the `EphemeralKeyReveal` query will not reveal the randomness used to compute the signature. Therefore, in this generic protocol construction, the security model we consider is slightly weaker than the actual $(\cdot)$AFL-eCK model, as it does not reveal the randomness used for signing with the `EphemeralKeyReveal` query. We name the weaker model as w$(\cdot)$AFL-eCK model. Later, in this chapter we will present different protocol construction, avoiding that additional restriction in the model.

5.3.4 Protocol Construction

In Table 5.1, we show the construction of protocol π. KeyGen, Enc and Dec are the key generation, encryption and decryption algorithms of the underlying CPLA2-secure (Section 2.4.3), $\epsilon -$ PG-IND-public-key cryptosystem PKE with ciphertext space $\widehat{\mathrm{C}}$. Moreover, we choose the message space M of the underlying public-key encryption scheme PKE to be equal to $\mathbb{Z}_q^*$. KG, Sign and Vfy are the key generation, signature generation and signature verification algorithms of the underlying leakage-resilient signature scheme SIG (Section 2.4.4). The protocol π is a Diffie-Hellman-type [DH76] key agreement protocol where $\mathbb{G}$ is a group of prime order q with generator g. After exchanging the public values both principals compute a Diffie-Hellman-type shared secret value, KDF is a secure key derivation function (Section 2.3.4) which generates a shared secret key (ms) using the Diffie-Hellman-type shared secret key, and PRF is a pseudo random function (Section 2.3.5) that is used to compute the session key using that shared key, ms,

and the protocol message sequence. The computations which leak information are underlined.

Remark 5.3.1. In Table 5.1, let $\widehat{\mathrm{C}}$ be the ciphertext space: in a setting like Naor and Segev [NS09], the random r values are not just chosen from C, but from $\widehat{\mathrm{C}} = \{0,1\}^{\mu} \times \mathrm{C} \times \{0,1\}^{t}$, which gives random $r \xleftarrow{\$} \widehat{\mathrm{C}}$ in the form (Ψ, c, seed).

A (Initiator)		B (Responder)
	Initial Setup	
$sk_A, vk_A \xleftarrow{\$} \mathrm{KG}(1^k)$		$sk_B, vk_B \xleftarrow{\$} \mathrm{KG}(1^k)$
$s_A, p_A \xleftarrow{\$} \mathrm{KeyGen}(1^k)$		$s_B, p_B \xleftarrow{\$} \mathrm{KeyGen}(1^k)$
	Protocol Execution	
$r_A \xleftarrow{\$} \widehat{\mathrm{C}}$		**If** $\mathrm{Vfy}(vk_A, X_A, \sigma_A) =$ "true" $\{$
$\widetilde{r_A} \leftarrow \underline{\mathrm{Dec}}(s_A, r_A)$		$r_B \xleftarrow{\$} \widehat{\mathrm{C}}$
$X_A \leftarrow g^{\widetilde{r_A}}$		$\widetilde{r_B} \leftarrow \underline{\mathrm{Dec}}(s_B, r_B)$
$\sigma_A \xleftarrow{\$} \underline{\mathrm{Sign}}(sk_A, (A, B, X_A))$	$\xrightarrow{A,B,X_A,\sigma_A}$	$X_B \leftarrow g^{\widetilde{r_B}}$
	$\xleftarrow{B,A,X_B,\sigma_B}$	$\sigma_B \xleftarrow{\$} \underline{\mathrm{Sign}}(sk_B, (B, A, X_B))$
If $\mathrm{Vfy}(vk_B, (B, A, X_B), \sigma_B) =$ "true" $\{$		
$\widetilde{r_A} \leftarrow \underline{\mathrm{Dec}}(s_A, r_A)$		
$ms \leftarrow \mathrm{KDF}(X_B^{\widetilde{r_A}}, \perp, k, \perp)$		$ms \leftarrow \mathrm{KDF}(X_A^{\widetilde{r_B}}, \perp, k, \perp)$
$K \leftarrow \mathrm{PRF}(ms, A\|\|X_A\|\|\sigma_A\|\|B\|\|X_B\|\|\sigma_B)$		$K \leftarrow \mathrm{PRF}(ms, A\|\|X_A\|\|\sigma_A\|\|B\|\|X_B\|\|\sigma_B)$
$\}$		$\}$
	K is the session key	

Table 5.1: Generic w(·)AFL-eCK-secure protocol construction: Protocol π

We found that the generic protocol construction presented in our initial publication of this work [ASB14], is vulnerable against active adversary. Following we explain the reason: In that protocol construction, the inputs to the key derivation function contain the Diffie-Hellman shared secret, and the identities of the initiator and the responder, A and B, respectively: $\mathrm{KDF}(g^{\widetilde{r_A}\widetilde{r_B}}, \perp, k, A||B)$. Assume that the target session is in B, if the adversary corrupts B and gets the signing key of B, re-sign the protocol message X_B computing the new signature σ'_B, that makes a different signature from σ_B, due to probabilistic signing algorithm. Then if the adversary sends B, A, X_B, σ'_B to B, instead of B, A, X_B, σ_B, and executes the rest of the protocol, it results that A's session and B's session are not matching, because the message B, A, X_B, σ_B computed by B is different from the message B, A, X_B, σ'_B received by A, but compute the same session key. Thus, the adversary can issue `SessionKeyReveal` query to the session at A and thus trivially learn the session key of the target session. Cremers [Cre11] showed that such attacks can be avoided by using the session identifier in the key derivation step together with other shared secrets. Thus, in this thesis we re-design the

protocol accordingly to ensure that mismatching sessions do not compute same session keys. Thus, the session key is derived using a pseudo random function as $K \leftarrow \text{PRF}(ms, A||X_A||\sigma_A||B||X_B||\sigma_B)$, such that it contains the session identifier $A||X_A||\sigma_A||B||X_B||\sigma_B$ as an input to the pseudo random function. The shared secret ms is derived as $ms \leftarrow \text{KDF}(X_B^{\widetilde{r_A}}, \perp, k, \perp)$. The σ input to the KDF is the Diffie-Hellman shared secret value $X_B^{\widetilde{r_A}}$, and it is a uniformly random element of the group, and therefore we can use KDF in the simulation without any problem.

5.3.5 Security Proof of the Protocol π in the w$(\cdot)$AFL-eCK Model

We prove the security of the generic protocol π in the w$(\cdot)$AFL-eCK model. If the underlying primitives are secure in the bounded or continuous leakage model, the protocol π is BAFL-eCK-secure or CAFL-eCK-secure respectively (with the restriction that `EphemeralKeyReveal` query does not reveal the randomness used in the signature computation).

Theorem 5.3.1. The protocol π is $\text{Adv}_{\pi}^{\boldsymbol{\lambda}-\text{w}(\cdot)\text{AFL-eCK}}$-secure, whenever the underlying public-key cryptosystem PKE is CPLA2-secure and $\epsilon - \text{PG-IND}$, the key derivation function KDF is secure with respect to an uniformly random source key material, the signature scheme SIG is UFCMLA-secure, PRF is a pseudo random function, the DDH and the ODH assumptions hold.

Let $\mathcal{U} = \{U_1, \ldots, U_{N_P}\}$ be a set of N_P parties. Each party U_i owns at most N_s number of protocol sessions. Let $\mathcal{A}$ be any PPT adversary against the protocol π. Then the advantage of $\mathcal{A}$ against $\boldsymbol{\lambda} - \text{w}(\cdot)\text{AFL-eCK}$-security of protocol π, $\text{Adv}_{\pi}^{\boldsymbol{\lambda}-\text{w}(\cdot)\text{AFL-eCK}}$ is:

$$\begin{aligned}\text{Adv}_{\pi}^{\boldsymbol{\lambda}-\text{w}(\cdot)\text{AFL-eCK}}(\mathcal{A}) \leq \max \Big[& N_P^2 N_s^2 \big[\big(\text{Adv}_{q,g}^{\text{DDH}}(\mathcal{C}) + \text{Adv}_{\text{KDF}}(\mathcal{B}) + \text{Adv}_{\text{PRF}}(\mathcal{J})\big) + \frac{1}{q}\big], \\ & N_P^2 N_s^2 \big[\big(\text{Adv}_{q,g}^{\text{DDH}}(\mathcal{C}) + \text{Adv}_{\text{KDF}}(\mathcal{B}) + \text{Adv}_{\text{PRF}}(\mathcal{J}) + 2\text{Adv}_{\text{PKE}}^{\text{CPLA2}}(\mathcal{D}) + 2\epsilon\big) + \frac{1}{q}\big], \\ & N_P^2 N_s^2 \big[\big(\text{Adv}_{q,g}^{\text{ODH}}(\mathcal{R}) + \text{Adv}_{\text{PRF}}(\mathcal{J}) + 2\text{Adv}_{\text{PKE}}^{\text{CPLA2}}(\mathcal{D}) + 2\epsilon\big) + \frac{1}{q}\big], N_P \text{Adv}_{\text{SIG}}^{\text{UFCMLA}}(\mathcal{E})\Big].\end{aligned}$$

where $\mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}, \mathcal{J}, \mathcal{R}$ are efficient algorithms constructed using the adversary $\mathcal{A}$, against the underlying key derivation function, KDF, DDH problem, public-key

cryptosystem, PKE, the signature scheme, SIG, pseudo random function, PRF, and Oracle Diffie-Hellman problem respectively. The PKE is $\epsilon - \text{PG-IND}$.

Proof. The proof is split into two main cases: when the partner to the test session exists, and when it does not.

Case 1: A partner session to the test session exists

In this case, the adversary is allowed to corrupt both principals or reveal ephemeral keys from both sessions. We assume that the adversary $\mathcal{A}$ can win the $\boldsymbol{\lambda} - \text{w}(\cdot)\text{AFL-eCK}$ challenge against the protocol π with advantage $\text{Adv}_{\pi}^{\boldsymbol{\lambda}-\text{w}(\cdot)\text{AFL-eCK}}(\mathcal{A})$. We split this case into four sub cases as follows:

1. Adversary corrupts both the owner and partner principals to the test session.
2. Adversary corrupts neither owner or nor partner principal to the test session.
3. Adversary corrupts the owner to the test session, but does not corrupt the partner to the test session.
4. Adversary corrupts the partner to the test session, but does not corrupt the owner to the test session.

Case 1.1: Adversary corrupts both the owner and partner principals to the test session

Game 1. This game is the original game. When the `Test` query is asked, the Game 1 challenger chooses a random bit $b \xleftarrow{\$} \{0, 1\}$. If $b = 1$, the real session key is given to $\mathcal{A}$, otherwise a random value chosen from the same session key space is given. This is the original game. Hence,

$$\text{Adv}_{\text{Game 1}}(\mathcal{A}) = \text{Adv}_{\pi,\text{Case 1.1}}^{\boldsymbol{\lambda}-\text{w}(\cdot)\text{AFL-eCK}}(\mathcal{A}). \tag{5.1}$$

Game 2. Abort the simulation if there exists two sessions outputting the same ephemeral public keys (Same $X = g^x$ values). Since the ephemeral keys are coming from $\mathbb{Z}_q^*$, the total number of ephemeral keys are q. Total number of session in the simulation is ${N_P}^2{N_s}^2$, because N_P is the number of protocol principals and each protocol principal owns N_s number of sessions. Hence,

$$|\mathrm{Adv}_{\text{Game }1}(\mathcal{A}) - \mathrm{Adv}_{\text{Game }2}(\mathcal{A})| \leq \frac{{N_P}^2 {N_s}^2}{q}. \tag{5.2}$$

Game 3. Before $\mathcal{A}$ begins, two distinct random principals $U^*, V^* \xleftarrow{\$} \{U_1, ..., U_{N_P}\}$ are chosen and two random numbers $s^*, t^* \xleftarrow{\$} \{1, ...N_s\}$ are chosen, where N_P is the number of protocol principals and N_s is the number of sessions on a principal. The session $\Pi^{s^*}_{U^*,V^*}$ is chosen as the target session and the session $\Pi^{t^*}_{V^*,U^*}$ is chosen as the partner to the target session. If the test session is not the session $\Pi^{s^*}_{U^*,V^*}$ or partner to the session is not $\Pi^{t^*}_{V^*,U^*}$, the Game 3 challenger aborts the game. Unless the incorrect choice happens, Game 3 is identical to Game 2. Hence,

$$\mathrm{Adv}_{\text{Game }3}(\mathcal{A}) = \frac{1}{{N_P}^2 N_s^2} \mathrm{Adv}_{\text{Game }2}(\mathcal{A}). \tag{5.3}$$

Game 4. Game 4 challenger randomly chooses $z \xleftarrow{\$} \mathbb{Z}_q^*$ and computes the session key of the target session and its partner session, using the KDF and the PRF as $ms \leftarrow \mathrm{KDF}(g^z, \perp, k, \perp)$ and $K \leftarrow \mathrm{PRF}(ms, U^*||X_{U^*}||\sigma_{U^*}||V^*||X_{V^*}||\sigma_{V^*})$, when U^* is the initiator, or $K \leftarrow \mathrm{PRF}(ms, V^*||X_{V^*}||\sigma_{V^*}||U^*||X_{U^*}||\sigma_{U^*})$, when U^* is the responder.

We construct an algorithm $\mathcal{C}$ against the DDH challenge, using the adversary $\mathcal{A}$. The DDH challenger sends values $(X = g^x, Y = g^y, Z = g^z)$ such that either $z = xy$ or $z \xleftarrow{\$} \mathbb{Z}_q^*$, as the inputs to the algorithm $\mathcal{C}$. $\mathcal{C}$ uses the value X as the ephemeral public key of U^* and Y as the ephemeral public key of V^* in the test session, and computes the session key using Z as the input to the KDF in the session key derivation process.

If $\mathcal{C}$'s input is a Diffie-Hellman triple, the simulation constructed by $\mathcal{C}$ is identical to Game 3, otherwise it is identical to Game 4. If $\mathcal{A}$ can distinguish whether $g^z = g^{xy}$ or not, then $\mathcal{C}$ can answer the DDH challenge. Note that `EphemeralKeyReveal`(U^*, V^*, s^*) or `EphemeralKeyReveal`(V^*, U^*, t^*) is prohibited since the adversary corrupts both the owner and the partner to the test session. $\mathcal{C}$ can answer all the adversarial queries allowed in this case, because it has all the long-term and ephemeral secret keys of the allowed queries. Hence,

$$|\mathrm{Adv}_{\text{Game }3}(\mathcal{A}) - \mathrm{Adv}_{\text{Game }4}(\mathcal{A})| \leq \mathrm{Adv}^{\mathrm{DDH}}_{q,g}(\mathcal{C}). \tag{5.4}$$

Game 5. The Game 5 challenger randomly chooses $ms \xleftarrow{\$} \{0,1\}^k$ and computes the session key of the target session and its partner session, using the PRF as $K \leftarrow \mathrm{PRF}(ms, U^*||X_{U^*}||\sigma_{U^*}||V^*||X_{V^*}||\sigma_{V^*})$, when U^* is the initiator or $K \leftarrow \mathrm{PRF}(ms, V^*||X_{V^*}||\sigma_{V^*}||U^*||X_{U^*}||\sigma_{U^*})$, when U^* is the responder.

We construct an algorithm $\mathcal{B}$ against a KDF challenger, using the adversary $\mathcal{A}$. The KDF challenger sends a ms value which is either generated using the KDF or randomly chosen. $\mathcal{B}$ uses the received ms value to compute the session key of the target session using the PRF.

If ms is computed using the KDF, simulation constructed by $\mathcal{B}$ is identical to Game 4, otherwise it is identical to Game 5. If $\mathcal{A}$ can distinguish the difference between Game 4 and Game 5, then $\mathcal{A}$ can be used as a subroutine of an algorithm $\mathcal{B}$, which is used to distinguish whether the ms value is computed using KDF or randomly chosen. $\mathcal{B}$ can answer all the adversarial queries allowed in this case, because it has all the long-term and ephemeral secret keys of the allowed queries. Hence,

$$|\mathrm{Adv}_{\mathrm{Game}\ 4}(\mathcal{A}) - \mathrm{Adv}_{\mathrm{Game}\ 5}(\mathcal{A})| \leq \mathrm{Adv}_{\mathrm{KDF}}(\mathcal{B}). \tag{5.5}$$

Game 6. The Game 6 challenger randomly chooses $K \xleftarrow{\$} \{0,1\}^k$ as session key of the target session and its partner session.

We construct an algorithm $\mathcal{J}$ against an Oracle$^{\mathrm{PRF}}$, using the adversary $\mathcal{A}$. The Oracle$^{\mathrm{PRF}}$ sends a K value which is either generated using the PRF with a hidden key, or a random function. $\mathcal{J}$ uses the received K as the session key of the target session.

If K is generated using the PRF with a hidden key, simulation constructed by $\mathcal{J}$ is identical to Game 5, otherwise it is identical to Game 6. If $\mathcal{A}$ can distinguish the difference between Game 5 and Game 6, then $\mathcal{A}$ can be used as a subroutine of an algorithm $\mathcal{J}$, which is used to distinguish whether the Oracle$^{\mathrm{PRF}}$ is real or a random function. $\mathcal{J}$ can answer all the adversarial queries allowed in this case, because it has all the long-term and ephemeral secret keys of the allowed queries. Hence,

$$|\mathrm{Adv}_{\mathrm{Game}\ 5}(\mathcal{A}) - \mathrm{Adv}_{\mathrm{Game}\ 6}(\mathcal{A})| \leq \mathrm{Adv}_{\mathrm{PRF}}(\mathcal{J}). \tag{5.6}$$

Semantic security of the session key in Game 6. Since the session key K of $\Pi^{s^*}_{U^*,V^*}$ is chosen randomly and independently from all other values, $\mathcal{A}$ does

not have any advantage in Game 6. Hence,

$$\mathrm{Adv}_{\text{Game 6}}(\mathcal{A}) = 0. \tag{5.7}$$

We find,

$$\mathrm{Adv}^{\boldsymbol{\lambda}-\mathrm{w}(\cdot)\text{AFL-eCK}}_{\pi,\text{Case 1.1}}(\mathcal{A}) \leq N_P^2 N_s^2 \Big[\big(\mathrm{Adv}^{\mathrm{DDH}}_{q,g}(\mathcal{C}) + \mathrm{Adv}_{\mathrm{KDF}}(\mathcal{B}) + \mathrm{Adv}_{\mathrm{PRF}}(\mathcal{J}) \big) + \frac{1}{q} \Big].$$

Case 1.2: Adversary corrupts neither owner or nor partner principal to the test session

Game 1. This game is the original game. When the `Test` query is asked, the Game 1 challenger chooses a random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 1$, the real session key is given to $\mathcal{A}$, otherwise a random value chosen from the same session key space is given. This is the original game. Hence,

$$\mathrm{Adv}_{\text{Game 1}}(\mathcal{A}) = \mathrm{Adv}^{\boldsymbol{\lambda}-\mathrm{w}(\cdot)\text{AFL-eCK}}_{\pi,\text{Case 1.2}}(\mathcal{A}). \tag{5.8}$$

Game 2. Abort the simulation if there exists two sessions outputting the same ephemeral public keys (Same $X = g^x$ values). Since the ephemeral keys are coming from $\mathbb{Z}_q^*$, the total number of ephemeral keys are q. Total number of session in the simulation is ${N_P}^2 {N_s}^2$, because N_P is the number of protocol principals and each protocol principal owns N_s number of sessions. Hence,

$$|\mathrm{Adv}_{\text{Game 1}}(\mathcal{A}) - \mathrm{Adv}_{\text{Game 2}}(\mathcal{A})| \leq \frac{{N_P}^2 {N_s}^2}{q}. \tag{5.9}$$

Game 3. Before $\mathcal{A}$ begins, two distinct random principals $U^*, V^* \xleftarrow{\$} \{U_1, ..., U_{N_P}\}$ are chosen and two random numbers $s^*, t^* \xleftarrow{\$} \{1, ...N_s\}$ are chosen, where N_P is the number of protocol principals and N_s is the number of sessions on a principal. The session $\Pi^{s^*}_{U^*,V^*}$ is chosen as the target session and the session $\Pi^{t^*}_{V^*,U^*}$ is chosen as the partner to the target session. If the test session is not the session $\Pi^{s^*}_{U^*,V^*}$ or partner to the session is not $\Pi^{t^*}_{V^*,U^*}$, the Game 3 challenger aborts the game. Unless the incorrect choice happens, Game 3 is identical to Game 2. Hence,

$$\mathrm{Adv}_{\text{Game 3}}(\mathcal{A}) = \frac{1}{{N_P}^2 N_s^2} \mathrm{Adv}_{\text{Game 2}}(\mathcal{A}). \tag{5.10}$$

Game 4. Game 4 challenger randomly chooses a pseudo-ephemeral value $\widetilde{r_{U^*}} \xleftarrow{\$} \mathbb{Z}_q^*$, and computes the ephemeral key $r_{U^*} \xleftarrow{\$} \text{Enc}(p_{U^*}, \widetilde{r_{U^*}})$, in the target session.

We introduce an algorithm $\mathcal{F}$ which is constructed using the adversary $\mathcal{A}$, against the ϵ-pair-generation indistinguishability challenger (ϵ-PG). $\mathcal{F}$ receives a pair $(r_{U^*}, \widetilde{r_{U^*}})$ such that $\widetilde{r_{U^*}} = \text{Dec}(s_{U^*}, r_{U^*})$. $\mathcal{F}$ uses r_{U^*} as the ephemeral key of U^* and $\widetilde{r_{U^*}}$ as the pseudo-ephemeral key of U^* in the target session.

If a random ephemeral key $r_{U^*} \xleftarrow{\$} \mathbb{Z}_q^*$ is chosen first and the pseudo-ephemeral value $\widetilde{r_{U^*}} \leftarrow \text{Dec}(s_{U^*}, r_{U^*})$ is computed, then the simulation constructed by $\mathcal{F}$ is identical to Game 3. Otherwise if a random pseudo-ephemeral value $\widetilde{r_{U^*}} \xleftarrow{\$} \mathbb{Z}_q^*$ is chosen first and the ephemeral key $r_{U^*} \xleftarrow{\$} \text{Enc}(p_{U^*}, \widetilde{r_{U^*}})$ is computed, then the simulation constructed by $\mathcal{F}$ is identical to Game 4. If $\mathcal{A}$ can distinguish the difference between Game 3 and Game 4, then $\mathcal{F}$ can distinguish whether a message/ciphertext pair (m, c) belongs to the distribution D_1 or D_2 (ϵ-pair-generation indistinguishability challenge). $\mathcal{F}$ can answer all the adversarial queries allowed in this case, because it has all the long-term and ephemeral secret keys of the allowed queries. Hence,

$$|\text{Adv}_{\text{Game 3}}(\mathcal{A}) - \text{Adv}_{\text{Game 4}}(\mathcal{A})| \leq \epsilon. \tag{5.11}$$

Game 5. Game 5 challenger randomly chooses a pseudo-ephemeral value $\widetilde{r'_{U^*}} \xleftarrow{\$} \mathbb{Z}_q^*$, and uses it as the pseudo ephemeral value of U^* in the target session.

We introduce an algorithm $\mathcal{D}$ which is constructed using the adversary $\mathcal{A}$, against the CPLA2 challenger. The algorithm $\mathcal{D}$ uses the public-key of the CPLA2 challenger as the public key of the protocol principal U^* and generates public/secret key pairs for all other protocol principals. $\mathcal{D}$ generates signing/verification key pairs for every protocol principal. $\mathcal{D}$ picks two random strings, $r_0, r_1 \xleftarrow{\$} \{0,1\}^k$ and passes them to the CPLA2 challenger. From the CPLA2 challenger, $\mathcal{D}$ receives a challenge ciphertext C_1 such that $C_1 \xleftarrow{\$} \text{Enc}(p_{U^*}, r_\theta)$ where $r_\theta = r_0$ or $r_\theta = r_1$. The following describes the procedure of answering queries:

- $\texttt{Send}(U, V, s, m, \mathbf{f})$ query: When $U = U^*$, $V = V^*$ and $s = s^*$, $\mathcal{D}$ takes r_1 as $\widetilde{r'_{U^*}}$, computes $g^{\widetilde{r'_{U^*}}}$ and computes its signature using the signing key sk_{U^*}. Then $\mathcal{D}$ creates the protocol message and sends it to $\mathcal{A}$ with the leakage $\mathbf{f}(s_{U^*})$, where the leakage $\mathbf{f}(s_{U^*})$ is obtained by accessing the leakage oracle of the CPLA2 challenger.

For all other `Send` queries, $\mathcal{D}$ can execute the protocol normally, because $\mathcal{D}$ has all the public keys and can compute protocol messages accordingly. Except U^* $\mathcal{D}$ can compute the leakage by its own, and for U^* $\mathcal{D}$ accesses the leakage oracle to obtain the leakage.

- `SessionKeyReveal`(U, V, s) query: $\mathcal{D}$ will abort if `SessionKeyReveal`(U^*, V^*, s^*) or `SessionKeyReveal`(V^*, U^*, t^*) query is asked. $\mathcal{D}$ can easily compute the answers using the corresponding psuedo-ephemeral keys for other `SessionKeyReveal` queries.

- `EphemeralKeyReveal`(U, V, s) query: For the `EphemeralKeyReveal`(U^*, V^*, s^*) query, $\mathcal{D}$ uses C_1 as the answer. For all other `EphemeralKeyReveal` queries $\mathcal{D}$ will answer with the corresponding ephemeral-key which is computed by encrypting a pseudo-ephemeral value with the secret key of the corresponding principal.

- `Corrupt`(U) query: Except for U^* and V^*, algorithm $\mathcal{D}$ can answer all other `Corrupt` queries. In this case we consider the situation in which the adversary corrupts neither owner or nor partner principal to the test session, so these exceptions will not occur.

- `Test`(U, s) query: The algorithm $\mathcal{D}$ will abort the game if the adversary issues a `Test` query other than `Test`(U^*, s^*). To compute the answer to the `Test`(U^*, s^*) query, the algorithm $\mathcal{D}$ computes:

  - If U^* is the initiator, computes $ms \leftarrow \mathrm{KDF}(X_{V^*}^{\widetilde{r'_{U^*}}}, \perp, k, \perp)$, $K \leftarrow \mathrm{PRF}(ms, U^*||X_{U^*}||\sigma_{U^*}||V^*||X_{V^*}||\sigma_{V^*})$ where $\widetilde{r'_{U^*}} = r_1$.
  - If U^* is the responder, computes $ms \leftarrow \mathrm{KDF}(X_{V^*}^{\widetilde{r'_{U^*}}}, \perp, k, \perp)$, $K \leftarrow \mathrm{PRF}(ms, V^*||X_{V^*}||\sigma_{V^*}||U^*||X_{U^*}||\sigma_{U^*})$.

  Then using K answers the `Test` query.

If $\theta = 1$, then r_1 is the decryption of C_1 and the simulation constructed by $\mathcal{D}$ is identical to Game 4 whereas if $\theta = 0$, then r_0 is the decryption of C_1 and the simulation constructed by $\mathcal{D}$ is identical to Game 5. If $\mathcal{A}$ can distinguish the difference between Game 4 and Game 5, then $\mathcal{D}$ can be used against a CPLA2 challenger. Hence,

$$|\mathrm{Adv}_{\text{Game }4}(\mathcal{A}) - \mathrm{Adv}_{\text{Game }5}(\mathcal{A})| \leq \mathrm{Adv}_{\mathrm{PKE}}^{\mathrm{CPLA2}}(\mathcal{D}). \tag{5.12}$$

Game 6. Game 6 challenger randomly chooses a pseudo-ephemeral value $\widetilde{r_{V^*}} \xleftarrow{\$} \mathbb{Z}_q^*$, and computes the ephemeral key $r_{V^*} \xleftarrow{\$} \text{Enc}(p_{V^*}, \widetilde{r_{V^*}})$ in the partner to the target session.

We introduce an algorithm $\mathcal{F}$ which is constructed using the adversary $\mathcal{A}$, against the ϵ-pair-generation indistinguishability challenger (ϵ-PG). $\mathcal{F}$ receives a pair $(r_{V^*}, \widetilde{r_{V^*}})$ such that $\widetilde{r_{V^*}} = \text{Dec}(s_{V^*}, r_{V^*})$. $\mathcal{F}$ uses r_{V^*} as the ephemeral key of V^* and $\widetilde{r_{V^*}}$ as the pseudo-ephemeral key of V^*.

If a random ephemeral key $r_{V^*} \xleftarrow{\$} \mathbb{Z}_q^*$ is chosen first and the pseudo-ephemeral value $\widetilde{r_{V^*}} \leftarrow \text{Dec}(s_{V^*}, r_{V^*})$ is computed, then the simulation constructed by $\mathcal{F}$ is identical to Game 5. Otherwise if a random pseudo-ephemeral value $\widetilde{r_{V^*}} \xleftarrow{\$} \mathbb{Z}_q^*$ is chosen first and the ephemeral key $r_{V^*} \xleftarrow{\$} \text{Enc}(p_{V^*}, \widetilde{r_{V^*}})$ is computed, then the simulation constructed by $\mathcal{F}$ is identical to Game 6. If $\mathcal{A}$ can distinguish the difference between Game 5 and Game 6, then $\mathcal{F}$ can distinguish whether a message/ciphertext pair (m, c) belongs to the distribution D_1 or D_2 (ϵ-pair-generation indistinguishability challenge). $\mathcal{F}$ can answer all the adversarial queries allowed in this case, because it has all the long-term and ephemeral secret keys of the allowed queries. Hence,

$$|\text{Adv}_{\text{Game 5}}(\mathcal{A}) - \text{Adv}_{\text{Game 6}}(\mathcal{A})| \leq \epsilon. \tag{5.13}$$

Game 7. Game 7 challenger randomly chooses a pseudo-ephemeral value $\widetilde{r'_{V^*}} \xleftarrow{\$} \mathbb{Z}_q^*$, and uses it as the pseudo ephemeral value of V^* in the partner to the target session.

We introduce an algorithm $\mathcal{D}$ which is constructed using the adversary $\mathcal{A}$, against the CPLA2 challenger. The algorithm $\mathcal{D}$ uses the public-key of the CPLA2 challenger as the public key of the protocol principal V^* and generates public/secret key pairs for all other protocol principals. $\mathcal{D}$ generates signing/verification key pairs for every protocol principal. $\mathcal{D}$ picks two random strings, $r'_0, r'_1 \xleftarrow{\$} \{0,1\}^k$ and passes them to the CPLA2 challenger. From the CPLA2 challenger, $\mathcal{D}$ receives a challenge ciphertext C_2 such that $C_2 \xleftarrow{\$} \text{Enc}(p_{V^*}, r'_\theta)$ where $r'_\theta = r'_0$ or $r'_\theta = r'_1$. The following describes the procedure of answering queries:

- $\texttt{Send}(U, V, s, m, \mathbf{f})$ query: When $U = V^*$, $V = U^*$ and $s = t^*$, $\mathcal{D}$ takes r'_1 as $\widetilde{r'_{V^*}}$, computes $g^{\widetilde{r'_{V^*}}}$ and computes its signature using the signing key sk_{V^*}. Then $\mathcal{D}$ creates the protocol message and sends it to $\mathcal{A}$ with the leakage

$\mathbf{f}(s_{V^*})$, where the leakage $\mathbf{f}(s_{V^*})$ is obtained by accessing the leakage oracle of the CPLA2 challenger.

For all other `Send` queries, $\mathcal{D}$ can execute the protocol normally, because $\mathcal{D}$ has all the public keys and can compute protocol messages accordingly. Except V^* $\mathcal{D}$ can compute the leakage by its own, and for V^* $\mathcal{D}$ accesses the leakage oracle to obtain the leakage.

- `SessionKeyReveal`(U, V, s) query: $\mathcal{D}$ will abort if `SessionKeyReveal`(U^*, V^*, s^*) or `SessionKeyReveal`(V^*, U^*, t^*) query is asked. $\mathcal{D}$ can easily compute the answers using the corresponding psuedo-ephemeral keys for other `SessionKeyReveal` queries.

- `EphemeralKeyReveal`(U, V, s) query: For the `EphemeralKeyReveal`(V^*, U^*, t^*) query, $\mathcal{D}$ uses C_2 as the answer. For all other `EphemeralKeyReveal` queries $\mathcal{D}$ will answer with the corresponding ephemeral-key which is computed by encrypting a pseudo-ephemeral value with the secret key of the corresponding principal.

- `Corrupt`(U) query: Except for U^* and V^*, algorithm $\mathcal{D}$ can answer all other `Corrupt` queries. In this case we consider the situation in which the adversary corrupts neither owner or nor partner principal to the test session, so these exceptions will not occur.

- `Test`(U, s) query: The algorithm $\mathcal{D}$ will abort the game if the adversary issues a `Test` query other than `Test`(U^*, s^*). To compute the answer to the `Test`(U^*, s^*) query, the algorithm $\mathcal{D}$ computes:

  - If U^* is the initiator, computes $ms \leftarrow \mathrm{KDF}(X_{U^*}^{\widetilde{r'_{V^*}}}, \perp, k, \perp)$, $K \leftarrow \mathrm{PRF}(ms, U^*||X_{U^*}||\sigma_{U^*}||V^*||X_{V^*}||\sigma_{V^*})$.
  - If U^* is the responder, $ms \leftarrow \mathrm{KDF}(X_{U^*}^{\widetilde{r'_{V^*}}}, \perp, k, \perp)$, $K \leftarrow \mathrm{PRF}(ms, V^*||X_{V^*}||\sigma_{V^*}||U^*||X_{U^*}||\sigma_{U^*})$.

  Then using K answers the `Test` query.

If $\theta = 1$, then r'_1 is the decryption of C_2 and the simulation constructed by $\mathcal{D}$ is identical to Game 6 whereas if $\theta = 0$, then r'_0 is the decryption of C_2 and the simulation constructed by $\mathcal{D}$ is identical to Game 7. If $\mathcal{A}$ can distinguish the

difference between Game 6 and Game 7, then $\mathcal{D}$ can be used against a CPLA2 challenger. Hence,

$$|\mathrm{Adv}_{\text{Game }6}(\mathcal{A}) - \mathrm{Adv}_{\text{Game }7}(\mathcal{A})| \leq \mathrm{Adv}^{\mathrm{CPLA2}}_{\mathrm{PKE}}(\mathcal{D}). \tag{5.14}$$

Game 8. Game 8 challenger randomly chooses $z \xleftarrow{\$} \mathbb{Z}_q^*$ and computes the session key of the target session and its partner session, using the KDF and the PRF as $ms \leftarrow \mathrm{KDF}(g^z, \perp, k, \perp)$ and $K \leftarrow \mathrm{PRF}(ms, U^*||X_{U^*}||\sigma_{U^*}||V^*||X_{V^*}||\sigma_{V^*})$, when U^* is the initiator or $K \leftarrow \mathrm{PRF}(ms, V^*||X_{V^*}||\sigma_{V^*}||U^*||X_{U^*}||\sigma_{U^*})$, when U^* is the responder.

We construct an algorithm $\mathcal{C}$ against the DDH challenge, using the adversary $\mathcal{A}$. The DDH challenger sends values $(X = g^x, Y = g^y, Z = g^z)$ such that either $z = xy$ or $z \xleftarrow{\$} \mathbb{Z}_q^*$, as the inputs to the algorithm $\mathcal{C}$. $\mathcal{C}$ uses the value X as the ephemeral public key of U^* and Y as the ephemeral public key of V^* in the test session, and computes the session key using Z as the input to the KDF in the session key derivation process.

If $\mathcal{C}$'s input is a Diffie-Hellman triple, the simulation constructed by $\mathcal{C}$ is identical to Game 7, otherwise it is identical to Game 8. If $\mathcal{A}$ can distinguish whether $g^z = g^{xy}$ or not, then $\mathcal{C}$ can answer the DDH challenge. $\mathcal{C}$ can answer all the adversarial queries allowed in this case, because it has all the long-term and ephemeral secret keys of the allowed queries. In this case `EphemeralKeyReveal` query is not allowed to the target session and its partner. Hence,

$$|\mathrm{Adv}_{\text{Game }7}(\mathcal{A}) - \mathrm{Adv}_{\text{Game }8}(\mathcal{A})| \leq \mathrm{Adv}^{\mathrm{DDH}}_{q,g}(\mathcal{C}). \tag{5.15}$$

Game 9. The Game 9 challenger randomly chooses $ms \xleftarrow{\$} \{0,1\}^k$ and computes the session key of the target session and its partner session, using the PRF as $K \leftarrow \mathrm{PRF}(ms, U^*||X_{U^*}||\sigma_{U^*}||V^*||X_{V^*}||\sigma_{V^*})$, when U^* is the initiator or $K \leftarrow \mathrm{PRF}(ms, V^*||X_{V^*}||\sigma_{V^*}||U^*||X_{U^*}||\sigma_{U^*})$, when U^* is the responder.

We construct an algorithm $\mathcal{B}$ against a KDF challenger, using the adversary $\mathcal{A}$. The KDF challenger sends a ms value which is either generated using the KDF or randomly chosen. $\mathcal{B}$ uses the received ms value to compute the session key of the target session using the PRF.

If ms is computed using the KDF, simulation constructed by $\mathcal{B}$ is identical to Game 8, otherwise it is identical to Game 9. If $\mathcal{A}$ can distinguish the difference between Game 8 and Game 9, then $\mathcal{A}$ can be used as a subroutine of an algorithm

$\mathcal{B}$, which is used to distinguish whether the ms value is computed using KDF or randomly chosen. $\mathcal{B}$ can answer all the adversarial queries allowed in this case, because it has all the long-term and ephemeral secret keys of the allowed queries. Hence,

$$|\mathrm{Adv}_{\text{Game }8}(\mathcal{A}) - \mathrm{Adv}_{\text{Game }9}(\mathcal{A})| \leq \mathrm{Adv}_{\mathrm{KDF}}(\mathcal{B}). \tag{5.16}$$

Game 10. The Game 10 challenger randomly chooses $K \xleftarrow{\$} \{0,1\}^k$ as session key of the target session and its partner session.

We construct an algorithm $\mathcal{J}$ against an Oracle$^{\mathrm{PRF}}$, using the adversary $\mathcal{A}$. The Oracle$^{\mathrm{PRF}}$ sends a K value which is either generated using the PRF with a hidden key, or a random function. $\mathcal{J}$ uses the received K as the session key of the target session.

If K is generated using the PRF with a hidden key, simulation constructed by $\mathcal{J}$ is identical to Game 9, otherwise it is identical to Game 10. If $\mathcal{A}$ can distinguish the difference between Game 9 and Game 10, then $\mathcal{A}$ can be used as a subroutine of an algorithm $\mathcal{J}$, which is used to distinguish whether the Oracle$^{\mathrm{PRF}}$ is real or a random function. $\mathcal{J}$ can answer all the adversarial queries allowed in this case, because it has all the long-term and ephemeral secret keys of the allowed queries. Hence,

$$|\mathrm{Adv}_{\text{Game }9}(\mathcal{A}) - \mathrm{Adv}_{\text{Game }10}(\mathcal{A})| \leq \mathrm{Adv}_{\mathrm{PRF}}(\mathcal{J}). \tag{5.17}$$

Semantic security of the session key in Game 10. Since the session key K of $\Pi^{s^*}_{U^*,V^*}$ is chosen randomly and independently from all other values, $\mathcal{A}$ does not have any advantage in Game 10. Hence,

$$\mathrm{Adv}_{\text{Game }10}(\mathcal{A}) = 0. \tag{5.18}$$

We find,

$$\mathrm{Adv}^{\boldsymbol{\lambda}-\mathrm{w}(\cdot)\mathrm{AFL\text{-}eCK}}_{\pi,\text{Case 1.2}}(\mathcal{A}) \leq N_P^2 N_s^2 \Big[\big(\mathrm{Adv}^{\mathrm{DDH}}_{q,g}(\mathcal{C}) + \mathrm{Adv}_{\mathrm{KDF}}(\mathcal{B}) + \mathrm{Adv}_{\mathrm{PRF}}(\mathcal{J}) + 2\mathrm{Adv}^{\mathrm{CPLA2}}_{\mathrm{PKE}}(\mathcal{D}) + 2\epsilon \big) + \frac{1}{q} \Big].$$

Case 1.3: Adversary corrupts the partner, but not the owner to the test session

Game 1. This game is the original game. When the `Test` query is asked, the Game 1 challenger chooses a random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 1$, the real session key is given to $\mathcal{A}$, otherwise a random value chosen from the same session key space is given. This is the original game. Hence,

$$\mathrm{Adv}_{\text{Game 1}}(\mathcal{A}) = \mathrm{Adv}^{\boldsymbol{\lambda}-\mathrm{w}(\cdot)\text{AFL-eCK}}_{\pi,\text{Case 1.3}}(\mathcal{A}). \tag{5.19}$$

Game 2. Abort the simulation if there exists two sessions outputting the same ephemeral public keys (Same $X = g^x$ values). Since the ephemeral keys are coming from $\mathbb{Z}_q^*$, the total number of ephemeral keys are q. Total number of session in the simulation is ${N_P}^2{N_s}^2$, because N_P is the number of protocol principals and each protocol principal owns N_s number of sessions. Hence,

$$|\mathrm{Adv}_{\text{Game 1}}(\mathcal{A}) - \mathrm{Adv}_{\text{Game 2}}(\mathcal{A})| \leq \frac{{N_P}^2{N_s}^2}{q}. \tag{5.20}$$

Game 3. Before $\mathcal{A}$ begins, two distinct random principals $U^*, V^* \xleftarrow{\$} \{U_1, ..., U_{N_P}\}$ are chosen and two random numbers $s^*, t^* \xleftarrow{\$} \{1, ...N_s\}$ are chosen, where N_P is the number of protocol principals and N_s is the number of sessions on a principal. The session $\Pi^{s^*}_{U^*,V^*}$ is chosen as the target session and the session $\Pi^{t^*}_{V^*,U^*}$ is chosen as the partner to the target session. If the test session is not the session $\Pi^{s^*}_{U^*,V^*}$ or partner to the session is not $\Pi^{t^*}_{V^*,U^*}$, the Game 3 challenger aborts the game. Unless the incorrect choice happens, Game 3 is identical to Game 2. Hence,

$$\mathrm{Adv}_{\text{Game 3}}(\mathcal{A}) = \frac{1}{{N_P}^2 N_s^2}\mathrm{Adv}_{\text{Game 2}}(\mathcal{A}). \tag{5.21}$$

Game 4. Game 4 challenger randomly chooses a pseudo-ephemeral value $\widetilde{r_{U^*}} \xleftarrow{\$} \mathbb{Z}_q^*$, and computes the ephemeral key $r_{U^*} \xleftarrow{\$} \mathrm{Enc}(p_{U^*}, \widetilde{r_{U^*}})$ in the target session.

We introduce an algorithm $\mathcal{F}$ which is constructed using the adversary $\mathcal{A}$, against the ϵ-pair-generation indistinguishability challenger (ϵ-PG). $\mathcal{F}$ receives a pair $(r_{U^*}, \widetilde{r_{U^*}})$ such that $\widetilde{r_{U^*}} = \mathrm{Dec}(s_{U^*}, r_{U^*})$. $\mathcal{F}$ uses r_{U^*} as the ephemeral key of U^* and $\widetilde{r_{U^*}}$ as the pseudo-ephemeral key of U^* in the target session.

If a random ephemeral key $r_{U^*} \xleftarrow{\$} \mathbb{Z}_q^*$ is chosen first and the pseudo-ephemeral value $\widetilde{r_{U^*}} \leftarrow \mathrm{Dec}(s_{U^*}, r_{U^*})$ is computed, then the simulation constructed by $\mathcal{F}$ is

identical to Game 3. Otherwise if a random pseudo-ephemeral value $\widetilde{r_{U^*}} \xleftarrow{\$} \mathbb{Z}_q^*$ is chosen first and the ephemeral key $r_{U^*} \xleftarrow{\$} \mathrm{Enc}(p_{U^*}, \widetilde{r_{U^*}})$ is computed, then the simulation constructed by $\mathcal{F}$ is identical to Game 4. If $\mathcal{A}$ can distinguish the difference between Game 3 and Game 4, then $\mathcal{F}$ can distinguish whether a message/ciphertext pair (m, c) belongs to the distribution D_1 or D_2 (ϵ-pair-generation indistinguishability challenge). $\mathcal{F}$ can answer all the adversarial queries allowed in this case, because it has all the long-term and ephemeral secret keys of the allowed queries. Hence,

$$|\mathrm{Adv}_{\text{Game 3}}(\mathcal{A}) - \mathrm{Adv}_{\text{Game 4}}(\mathcal{A})| \leq \epsilon. \tag{5.22}$$

Game 5. Game 5 challenger randomly chooses a pseudo-ephemeral value $\widetilde{r'_{U^*}} \xleftarrow{\$} \mathbb{Z}_q^*$, and uses it as the pseudo ephemeral value of U^* in the target session.

We introduce an algorithm $\mathcal{D}$ which is constructed using the adversary $\mathcal{A}$, against the CPLA2 challenger. The algorithm $\mathcal{D}$ uses the public-key of the CPLA2 challenger as the public key of the protocol principal U^* and generates public/secret key pairs for all other protocol principals. $\mathcal{D}$ generates signing/verification key pairs for every protocol principal. $\mathcal{D}$ picks two random strings, $r_0, r_1 \xleftarrow{\$} \{0,1\}^k$ and passes them to the CPLA2 challenger. From the CPLA2 challenger, $\mathcal{D}$ receives a challenge ciphertext C_1 such that $C_1 \xleftarrow{\$} \mathrm{Enc}(p_{U^*}, r_\theta)$ where $r_\theta = r_0$ or $r_\theta = r_1$. The following describes the procedure of answering queries:

- `Send`$(U, V, s, m, \mathbf{f})$ query: When $U = U^*$, $V = V^*$ and $s = s^*$, $\mathcal{D}$ takes r_1 as $\widetilde{r'_{U^*}}$, computes $g^{\widetilde{r'_{U^*}}}$ and computes its signature using the signing key sk_{U^*}. Then $\mathcal{D}$ creates the protocol message and sends it to $\mathcal{A}$ with the leakage $\mathbf{f}(s_{U^*})$, where the leakage $\mathbf{f}(s_{U^*})$ is obtained by accessing the leakage oracle of the CPLA2 challenger.

  For all other `Send` queries, $\mathcal{D}$ can execute the protocol normally, because $\mathcal{D}$ has all the public keys and can compute protocol messages accordingly. Except U^* $\mathcal{D}$ can compute the leakage by its own, and for U^* $\mathcal{D}$ accesses the leakage oracle to obtain the leakage.

- `SessionKeyReveal`(U, V, s) query: $\mathcal{D}$ will abort if `SessionKeyReveal`(U^*, V^*, s^*) or `SessionKeyReveal`(V^*, U^*, t^*) query is asked. $\mathcal{D}$ can easily compute the answers using the corresponding psuedo-ephemeral keys for other

`SessionKeyReveal` queries.

- `EphemeralKeyReveal`(U, V, s) query: For the `EphemeralKeyReveal`(U^*, V^*, s^*) query, $\mathcal{D}$ uses C_1 as the answer. For all other `EphemeralKeyReveal` queries $\mathcal{D}$ will answer with the corresponding ephemeral-key which is computed by encrypting a pseudo-ephemeral value with the secret key of the corresponding principal.

- `Corrupt`(U) query: Except for U^*, algorithm $\mathcal{D}$ can answer all other `Corrupt` queries. In this case we consider the situation in which the adversary does not corrupt the partner to the test session, so these exceptions will not occur.

- `Test`(U, s) query: The algorithm $\mathcal{D}$ will abort the game if the adversary issues a `Test` query other than `Test`(U^*, s^*). To compute the answer to the `Test`(U^*, s^*) query, the algorithm $\mathcal{D}$ computes:

  - If U^* is the initiator, computes $ms \leftarrow \mathrm{KDF}(\widetilde{X^{r'_{U^*}}_{V^*}}, \perp, k, \perp)$, $K \leftarrow \mathrm{PRF}(ms, U^*||X_{U^*}||\sigma_{U^*}||V^*||X_{V^*}||\sigma_{V^*})$.
  - If U^* is the responder, computes $ms \leftarrow \mathrm{KDF}(\widetilde{X^{r'_{U^*}}_{V^*}}, \perp, k, \perp)$, $K \leftarrow \mathrm{PRF}(ms, V^*||X_{V^*}||\sigma_{V^*}||U^*||X_{U^*}||\sigma_{U^*})$.

  Then using K answers the `Test` query.

If $\theta = 1$, then r_1 is the decryption of C_1 and the simulation constructed by $\mathcal{D}$ is identical to Game 4 whereas if $\theta = 0$, then r_0 is the decryption of C_1 and the simulation constructed by $\mathcal{D}$ is identical to Game 5. If $\mathcal{A}$ can distinguish the difference between Game 4 and Game 5, then $\mathcal{D}$ can be used against a CPLA2 challenger. Hence,

$$|\mathrm{Adv}_{\text{Game }4}(\mathcal{A}) - \mathrm{Adv}_{\text{Game }5}(\mathcal{A})| \leq \mathrm{Adv}^{\mathrm{CPLA2}}_{\mathrm{PKE}}(\mathcal{D}). \tag{5.23}$$

Game 6. Game 6 challenger randomly chooses $z \xleftarrow{\$} \mathbb{Z}^*_q$ and computes the session key of the target session and its partner session, using the KDF and the PRF as $ms \leftarrow \mathrm{KDF}(g^z, \perp, k, \perp)$ and $K \leftarrow \mathrm{PRF}(ms, U^*||X_{U^*}||\sigma_{U^*}||V^*||X_{V^*}||\sigma_{V^*})$, when U^* is the initiator, or $K \leftarrow \mathrm{PRF}(ms, V^*||X_{V^*}||\sigma_{V^*}||U^*||X_{U^*}||\sigma_{U^*})$, when U^* is the responder.

We construct an algorithm $\mathcal{C}$ against the DDH challenge, using the adversary $\mathcal{A}$. The DDH challenger sends values $(X = g^x, Y = g^y, Z = g^z)$ such that either $z = xy$ or $z \xleftarrow{\$} \mathbb{Z}_q^*$, as the inputs to the algorithm $\mathcal{C}$. $\mathcal{C}$ uses the value X as the ephemeral public key of U^* and Y as the ephemeral public key of V^* in the test session, and computes the session key using Z as the input to the KDF in the session key derivation process.

If $\mathcal{C}$'s input is a Diffie-Hellman triple, the simulation constructed by $\mathcal{C}$ is identical to Game 5, otherwise it is identical to Game 6. If $\mathcal{A}$ can distinguish whether $g^z = g^{xy}$ or not, then $\mathcal{C}$ can answer the DDH challenge. $\mathcal{C}$ can answer all the adversarial queries allowed in this case, because it has all the long-term and ephemeral secret keys of the allowed queries. Hence,

$$|\mathrm{Adv}_{\text{Game 5}}(\mathcal{A}) - \mathrm{Adv}_{\text{Game 6}}(\mathcal{A})| \leq \mathrm{Adv}_{q,g}^{\mathrm{DDH}}(\mathcal{C}). \tag{5.24}$$

Game 7. The Game 7 challenger randomly chooses $ms \xleftarrow{\$} \{0,1\}^k$ and computes the session key of the target session and its partner session, using the PRF as $K \leftarrow \mathrm{PRF}(ms, U^*||X_{U^*}||\sigma_{U^*}||V^*||X_{V^*}||\sigma_{V^*})$, when U^* is the initiator or $K \leftarrow \mathrm{PRF}(ms, V^*||X_{V^*}||\sigma_{V^*}||U^*||X_{U^*}||\sigma_{U^*})$, when U^* is the responder.

We construct an algorithm $\mathcal{B}$ against a KDF challenger, using the adversary $\mathcal{A}$. The KDF challenger sends a ms value which is either generated using the KDF or randomly chosen. $\mathcal{B}$ uses the received ms value to compute the session key of the target session using the PRF.

If ms is computed using the KDF, simulation constructed by $\mathcal{B}$ is identical to Game 6, otherwise it is identical to Game 7. If $\mathcal{A}$ can distinguish the difference between Game 6 and Game 7, then $\mathcal{A}$ can be used as a subroutine of an algorithm $\mathcal{B}$, which is used to distinguish whether the ms value is computed using KDF or randomly chosen. $\mathcal{B}$ can answer all the adversarial queries allowed in this case, because it has all the long-term and ephemeral secret keys of the allowed queries. Hence,

$$|\mathrm{Adv}_{\text{Game 6}}(\mathcal{A}) - \mathrm{Adv}_{\text{Game 7}}(\mathcal{A})| \leq \mathrm{Adv}_{\mathrm{KDF}}(\mathcal{B}). \tag{5.25}$$

Game 8. The Game 8 challenger randomly chooses $K \xleftarrow{\$} \{0,1\}^k$ as session key of the target session and its partner session.

We construct an algorithm $\mathcal{J}$ against an $\mathrm{Oracle}^{\mathrm{PRF}}$, using the adversary $\mathcal{A}$. The $\mathrm{Oracle}^{\mathrm{PRF}}$ sends a K value which is either generated using the PRF with a

hidden key, or a random function. $\mathcal{J}$ uses the received K as the session key of the target session.

If K is generated using the PRF with a hidden key, simulation constructed by $\mathcal{J}$ is identical to Game 7, otherwise it is identical to Game 8. If $\mathcal{A}$ can distinguish the difference between Game 7 and Game 8, then $\mathcal{A}$ can be used as a subroutine of an algorithm $\mathcal{J}$, which is used to distinguish whether the $\text{Oracle}^{\text{PRF}}$ is real or a random function. $\mathcal{J}$ can answer all the adversarial queries allowed in this case, because it has all the long-term and ephemeral secret keys of the allowed queries. Hence,

$$|\text{Adv}_{\text{Game }7}(\mathcal{A}) - \text{Adv}_{\text{Game }8}(\mathcal{A})| \leq \text{Adv}_{\text{PRF}}(\mathcal{J}). \tag{5.26}$$

Semantic security of the session key in Game 8. Since the session key K of $\Pi^{s^*}_{U^*,V^*}$ is chosen randomly and independently from all other values, $\mathcal{A}$ does not have any advantage in Game 8. Hence,

$$\text{Adv}_{\text{Game }8}(\mathcal{A}) = 0. \tag{5.27}$$

We find,

$$\text{Adv}^{\boldsymbol{\lambda}-\text{w}(\cdot)\text{AFL-eCK}}_{\pi,\text{Case 1.3}}(\mathcal{A}) \leq N_P^2 N_s^2 \Big[\big(\text{Adv}^{\text{DDH}}_{q,g}(\mathcal{C}) + \text{Adv}_{\text{KDF}}(\mathcal{B}) + \text{Adv}_{\text{PRF}}(\mathcal{J}) + \text{Adv}^{\text{CPLA2}}_{\text{PKE}}(\mathcal{D}) + \epsilon \big) + \frac{1}{q} \Big].$$

Case 1.4: Adversary corrupts the owner, but not the partner to the test session

Game 1. This game is the original game. When the `Test` query is asked, the Game 1 challenger chooses a random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 1$, the real session key is given to $\mathcal{A}$, otherwise a random value chosen from the same session key space is given. This is the original game. Hence,

$$\text{Adv}_{\text{Game }1}(\mathcal{A}) = \text{Adv}^{\boldsymbol{\lambda}-\text{w}(\cdot)\text{AFL-eCK}}_{\pi,\text{Case 1.4}}(\mathcal{A}). \tag{5.28}$$

Game 2. Abort the simulation if there exists two sessions outputting the same ephemeral public keys (Same $X = g^x$ values). Since the ephemeral keys are coming from $\mathbb{Z}_q^*$, the total number of ephemeral keys are q. Total number of session in

the simulation is ${N_P}^2{N_s}^2$, because N_P is the number of protocol principals and each protocol principal owns N_s number of sessions. Hence,

$$|\mathrm{Adv}_{\text{Game }1}(\mathcal{A}) - \mathrm{Adv}_{\text{Game }2}(\mathcal{A})| \leq \frac{{N_P}^2{N_s}^2}{q}. \tag{5.29}$$

Game 3. Before $\mathcal{A}$ begins, two distinct random principals $U^*, V^* \xleftarrow{\$} \{U_1, ..., U_{N_P}\}$ are chosen and two random numbers $s^*, t^* \xleftarrow{\$} \{1, ...N_s\}$ are chosen, where N_P is the number of protocol principals and N_s is the number of sessions on a principal. The session $\Pi^{s^*}_{U^*,V^*}$ is chosen as the target session and the session $\Pi^{t^*}_{V^*,U^*}$ is chosen as the partner to the target session. If the test session is not the session $\Pi^{s^*}_{U^*,V^*}$ or partner to the session is not $\Pi^{t^*}_{V^*,U^*}$, the Game 3 challenger aborts the game. Unless the incorrect choice happens, Game 3 is identical to Game 2. Hence,

$$\mathrm{Adv}_{\text{Game }3}(\mathcal{A}) = \frac{1}{{N_P}^2 N_s^2}\mathrm{Adv}_{\text{Game }2}(\mathcal{A}). \tag{5.30}$$

Game 4. Game 4 challenger randomly chooses a pseudo-ephemeral value $\widetilde{r_{V^*}} \xleftarrow{\$} \mathbb{Z}_q^*$, and computes the ephemeral key $r_{V^*} \xleftarrow{\$} \mathrm{Enc}(p_{V^*}, \widetilde{r_{V^*}})$, in the partner to the target session.

We introduce an algorithm $\mathcal{F}$ which is constructed using the adversary $\mathcal{A}$, against the ϵ-pair-generation indistinguishability challenger (ϵ-PG). $\mathcal{F}$ receives a pair $(r_{V^*}, \widetilde{r_{V^*}})$ such that $\widetilde{r_{V^*}} = \mathrm{Dec}(s_{V^*}, r_{V^*})$. $\mathcal{F}$ uses r_{V^*} as the ephemeral key of V^* and $\widetilde{r_{V^*}}$ as the pseudo-ephemeral key of V^* in the partner to the target session.

If a random ephemeral key $r_{V^*} \xleftarrow{\$} \mathbb{Z}_q^*$ is chosen first and the pseudo-ephemeral value $\widetilde{r_{V^*}} \leftarrow \mathrm{Dec}(s_{V^*}, r_{V^*})$ is computed, then the simulation constructed by $\mathcal{F}$ is identical to Game 3. Otherwise if a random pseudo-ephemeral value $\widetilde{r_{V^*}} \xleftarrow{\$} \mathbb{Z}_q^*$ is chosen first and the ephemeral key $r_{V^*} \xleftarrow{\$} \mathrm{Enc}(p_{V^*}, \widetilde{r_{V^*}})$ is computed, then the simulation constructed by $\mathcal{F}$ is identical to Game 4. If $\mathcal{A}$ can distinguish the difference between Game 3 and Game 4, then $\mathcal{F}$ can distinguish whether a message/ciphertext pair (m, c) belongs to the distribution D_1 or D_2 (ϵ-pair-generation indistinguishability challenge). $\mathcal{F}$ can answer all the adversarial queries allowed in this case, because it has all the long-term and ephemeral secret keys of the allowed queries. Hence,

$$|\mathrm{Adv}_{\text{Game }3}(\mathcal{A}) - \mathrm{Adv}_{\text{Game }4}(\mathcal{A})| \leq \epsilon. \tag{5.31}$$

Game 5. Game 5 challenger randomly chooses a pseudo-ephemeral value $\widetilde{r'_{V^*}} \xleftarrow{\$} \mathbb{Z}_q^*$, and uses it as the pseudo ephemeral value of V^* in the partner to the target session.

We introduce an algorithm $\mathcal{D}$ which is constructed using the adversary $\mathcal{A}$, against the CPLA2 challenger. The algorithm $\mathcal{D}$ uses the public-key of the CPLA2 challenger as the public key of the protocol principal V^* and generates public/secret key pairs for all other protocol principals. $\mathcal{D}$ generates signing/verification key pairs for every protocol principal. $\mathcal{D}$ picks two random strings, $r'_0, r'_1 \xleftarrow{\$} \{0,1\}^k$ and passes them to the CPLA2 challenger. From the CPLA2 challenger, $\mathcal{D}$ receives a challenge ciphertext C_2 such that $C_2 \xleftarrow{\$} \mathrm{Enc}(p_{V^*}, r'_\theta)$ where $r'_\theta = r'_0$ or $r'_\theta = r'_1$. The following describes the procedure of answering queries:

- `Send`$(U, V, s, m, \mathbf{f})$ query: When $U = V^*$, $V = U^*$ and $s = t^*$, $\mathcal{D}$ takes r'_1 as $\widetilde{r'_{V^*}}$, computes $g^{\widetilde{r'_{V^*}}}$ and computes its signature using the signing key sk_{V^*}. Then $\mathcal{D}$ creates the protocol message and sends it to $\mathcal{A}$ with the leakage $\mathbf{f}(s_{V^*})$, where the leakage $\mathbf{f}(s_{V^*})$ is obtained by accessing the leakage oracle of the CPLA2 challenger.

  For all other `Send` queries, $\mathcal{D}$ can execute the protocol normally, because $\mathcal{D}$ has all the public keys and can compute protocol messages accordingly. Except V^* $\mathcal{D}$ can compute the leakage by its own, and for V^* $\mathcal{D}$ accesses the leakage oracle to obtain the leakage.

- `SessionKeyReveal`(U, V, s) query: $\mathcal{D}$ will abort if `SessionKeyReveal`(U^*, V^*, s^*) or `SessionKeyReveal`(V^*, U^*, t^*) query is asked. $\mathcal{D}$ can easily compute the answers using the corresponding psuedo-ephemeral keys for other `SessionKeyReveal` queries.

- `EphemeralKeyReveal`(U, V, s) query: For the `EphemeralKeyReveal`(V^*, U^*, t^*) query, $\mathcal{D}$ uses C_2 as the answer. For all other `EphemeralKeyReveal` queries $\mathcal{D}$ will answer with the corresponding ephemeral-key which is computed by encrypting a pseudo-ephemeral value with the secret key of the corresponding principal.

- `Corrupt`(U) query: Except for U^* and V^*, algorithm $\mathcal{D}$ can answer all other `Corrupt` queries. In this case we consider the situation in which the

adversary does not corrupt the partner principal to the test session, so these exceptions will not occur.

- $\texttt{Test}(U, s)$ query: The algorithm $\mathcal{D}$ will abort the game if the adversary issues a `Test` query other than $\texttt{Test}(U^*, s^*)$. To compute the answer to the $\texttt{Test}(U^*, s^*)$ query, the algorithm $\mathcal{D}$ computes:
  - If U^* is the initiator, $ms \leftarrow \text{KDF}(\widetilde{X_{U^*}^{r'_{V^*}}}, \perp, k, \perp)$, $K \leftarrow \text{PRF}(ms, U^* ||X_{U^*}||\sigma_{U^*}||V^*||X_{V^*}||\sigma_{V^*})$ where $\widetilde{r'_{U^*}} = r_1$..
  - If U^* is the responder computes $ms \leftarrow \text{KDF}(\widetilde{X_{U^*}^{r'_{V^*}}}, \perp, k, \perp)$, $K \leftarrow \text{PRF}(ms, V^*||X_{V^*}||\sigma_{V^*}||U^*||X_{U^*}||\sigma_{U^*})$.

  Then using K answers the `Test` query.

If $\theta = 1$, then r'_1 is the decryption of C_2 and the simulation constructed by $\mathcal{D}$ is identical to Game 4 whereas if $\theta = 0$, then r'_0 is the decryption of C_2 and the simulation constructed by $\mathcal{D}$ is identical to Game 5. If $\mathcal{A}$ can distinguish the difference between Game 4 and Game 5, then $\mathcal{D}$ can be used against a CPLA2 challenger. Hence,

$$|\text{Adv}_{\text{Game 4}}(\mathcal{A}) - \text{Adv}_{\text{Game 5}}(\mathcal{A})| \leq \text{Adv}_{\text{PKE}}^{\text{CPLA2}}(\mathcal{D}). \tag{5.32}$$

Game 6. Game 6 challenger randomly chooses $z \xleftarrow{\$} \mathbb{Z}_q^*$ and computes the session key of the target session and its partner session, using the KDF and the PRF as $ms \leftarrow \text{KDF}(g^z, \perp, k, \perp)$ and $K \leftarrow \text{PRF}(ms, U^*||X_{U^*}||\sigma_{U^*}||V^*||X_{V^*}||\sigma_{V^*})$, when U^* is the initiator, or $K \leftarrow \text{PRF}(ms, V^*||X_{V^*}||\sigma_{V^*}||U^*||X_{U^*}||\sigma_{U^*})$, when U^* is the responder.

We construct an algorithm $\mathcal{C}$ against the DDH challenge, using the adversary $\mathcal{A}$. The DDH challenger sends values $(X = g^x, Y = g^y, Z = g^z)$ such that either $z = xy$ or $z \xleftarrow{\$} \mathbb{Z}_q^*$, as the inputs to the algorithm $\mathcal{C}$. $\mathcal{C}$ uses the value X as the ephemeral public key of U^* and Y as the ephemeral public key of V^* in the test session, and computes the session key using Z as the input to the KDF in the session key derivation process.

If $\mathcal{C}$'s input is a Diffie-Hellman triple, the simulation constructed by $\mathcal{C}$ is identical to Game 5, otherwise it is identical to Game 6. If $\mathcal{A}$ can distinguish whether $g^z = g^{xy}$ or not, then $\mathcal{C}$ can answer the DDH challenge. $\mathcal{C}$ can answer all

the adversarial queries allowed in this case, because it has all the long-term and ephemeral secret keys of the allowed queries. Hence,

$$|\mathrm{Adv}_{\text{Game }5}(\mathcal{A}) - \mathrm{Adv}_{\text{Game }6}(\mathcal{A})| \leq \mathrm{Adv}^{\mathrm{DDH}}_{q,g}(\mathcal{C}). \tag{5.33}$$

Game 7. The Game 7 challenger randomly chooses $ms \xleftarrow{\$} \{0,1\}^k$ and computes the session key of the target session and its partner session, using the PRF as $K \leftarrow \mathrm{PRF}(ms, U^*||X_{U^*}||\sigma_{U^*}||V^*||X_{V^*}||\sigma_{V^*})$, when U^* is the initiator or $K \leftarrow \mathrm{PRF}(ms, V^*||X_{V^*}||\sigma_{V^*}||U^*||X_{U^*}||\sigma_{U^*})$, when U^* is the responder.

We construct an algorithm $\mathcal{B}$ against a KDF challenger, using the adversary $\mathcal{A}$. The KDF challenger sends a ms value which is either generated using the KDF or randomly chosen. $\mathcal{B}$ uses the received ms value to compute the session key of the target session using the PRF.

If ms is computed using the KDF, simulation constructed by $\mathcal{B}$ is identical to Game 6, otherwise it is identical to Game 7. If $\mathcal{A}$ can distinguish the difference between Game 6 and Game 7, then $\mathcal{A}$ can be used as a subroutine of an algorithm $\mathcal{B}$, which is used to distinguish whether the ms value is computed using KDF or randomly chosen. $\mathcal{B}$ can answer all the adversarial queries allowed in this case, because it has all the long-term and ephemeral secret keys of the allowed queries. Hence,

$$|\mathrm{Adv}_{\text{Game }6}(\mathcal{A}) - \mathrm{Adv}_{\text{Game }7}(\mathcal{A})| \leq \mathrm{Adv}_{\mathrm{KDF}}(\mathcal{B}). \tag{5.34}$$

Game 8. The Game 8 challenger randomly chooses $K \xleftarrow{\$} \{0,1\}^k$ as session key of the target session and its partner session.

We construct an algorithm $\mathcal{J}$ against an Oracle$^{\mathrm{PRF}}$, using the adversary $\mathcal{A}$. The Oracle$^{\mathrm{PRF}}$ sends a K value which is either generated using the PRF with a hidden key, or a random function. $\mathcal{J}$ uses the received K as the session key of the target session.

If K is generated using the PRF with a hidden key, simulation constructed by $\mathcal{J}$ is identical to Game 7, otherwise it is identical to Game 8. If $\mathcal{A}$ can distinguish the difference between Game 7 and Game 8, then $\mathcal{A}$ can be used as a subroutine of an algorithm $\mathcal{J}$, which is used to distinguish whether the Oracle$^{\mathrm{PRF}}$ is real or a random function. $\mathcal{J}$ can answer all the adversarial queries allowed in this case, because it has all the long-term and ephemeral secret keys of the allowed queries. Hence,

$$|\mathrm{Adv}_{\text{Game }7}(\mathcal{A}) - \mathrm{Adv}_{\text{Game }8}(\mathcal{A})| \leq \mathrm{Adv}_{\mathrm{PRF}}(\mathcal{J}). \tag{5.35}$$

Semantic security of the session key in Game 8. Since the session key K of $\Pi^{s^*}_{U^*,V^*}$ is chosen randomly and independently from all other values, $\mathcal{A}$ does not have any advantage in Game 8. Hence,

$$\mathrm{Adv}_{\text{Game }8}(\mathcal{A}) = 0. \tag{5.36}$$

We find,

$$\mathrm{Adv}^{\boldsymbol{\lambda}-\mathrm{w}(\cdot)\mathrm{AFL\text{-}eCK}}_{\pi,\text{Case 1.4}}(\mathcal{A}) \leq N_P^2 N_s^2 \Big[\big(\mathrm{Adv}^{\mathrm{DDH}}_{q,g}(\mathcal{C}) + \mathrm{Adv}_{\mathrm{KDF}}(\mathcal{B}) + \mathrm{Adv}_{\mathrm{PRF}}(\mathcal{J}) + \mathrm{Adv}^{\mathrm{CPLA2}}_{\mathrm{PKE}}(\mathcal{D}) + \epsilon \big) + \frac{1}{q} \Big].$$

Case 2: A partner session to the test session does not exist

When the partner session does not exist, the owner of the test session shares the session key with the active adversary. In this situation adversary is not allowed to corrupt the intended partner principal to the test session. We split this case into two sub cases as follows:

1. Test session is at the responder.
2. Test session is at the initiator.

Case 2.1: Test session is at the responder Let V^* be the initiator and U^* be the responder. Let X_{U^*} be the ephemeral public key of U^*, and X_{V^*} be the ephemeral public key of V^*, in the target session. In this case there are three sub cases, which address the three different situations occur when the challenger interacts with the adversary. We will analyze the adversaries advantage in winning the w(·)AFL-eCK challenge in following three different cases.

- (a) There is no session at peer V^* with X_{V^*}: Here the adversary tries to compute the protocol message from V^* to U^*, by its own.
- (b) There exists a session at V^* with X_{V^*} and X_{U^*} (but σ_{U^*} computed by U^* is different from the σ_{U^*} received to V^* with the protocol message, in the target session): For instance the adversary corrupts U^*, re-sign the protocol

message from U^* to V^*, executes the protocol and makes a non-matching session at V^*. Then, tries to reveal the session key of that non-matching session at V^*, and win the game.

- (c) There exists a session at V^* with X_{V^*} and $X'_{U^*} \neq X_{U^*}$: Here the adversary, changes the message from U^* to V^*, such that there is no matching session at V^*.

Case 2.1.a: There is no session at V^* with X_{V^*}

Assume that the adversary $\mathcal{A}$ asks a `Send` query to some fresh session, such that it accepts, but the signature used in the query is not generated by a legitimate party.

Game 1. This game is the original game. Hence,

$$\mathrm{Adv}_{\text{Game 1}}(\mathcal{A}) = \mathrm{Adv}^{\boldsymbol{\lambda}-\mathrm{w}(\cdot)\mathrm{AFL\text{-}eCK}}_{\pi,\text{Case 2.1.a}}(\mathcal{A}). \tag{5.37}$$

Game 2. Before $\mathcal{A}$ begins, the Game 2 challenger guesses the identity, V^*, of the partner principal to the test session and if the guess is incorrect it aborts the game. The probability of Game 2 to be aborted due to incorrect guess of the partner principal to the test session is $1 - \frac{1}{N_P}$. Unless the incorrect guess happens, Game 2 is identical to Game1. Hence,

$$\mathrm{Adv}_{\text{Game 2}}(\mathcal{A}) = \frac{1}{N_P}\mathrm{Adv}_{\text{Game 1}}(\mathcal{A}). \tag{5.38}$$

The algorithm $\mathcal{E}$ sets the verification key of the signature scheme challenger to the principal V^*. The owner principal accepts the message coming from the intended partner, because the owner computes $\mathrm{Vfy}(vk_{V^*}, X_{V^*}, \sigma_{V^*})$ is "true". But the principal V^* is not corrupted and the message X_{V^*} is not signed by the principal V^*, because there is no partner to the test session. Hence,

$$\mathrm{Adv}_{\text{Game 2}}(\mathcal{A}) = \mathrm{Adv}^{\mathrm{UFCMLA}}_{\mathrm{SIG}}(\mathcal{E}). \tag{5.39}$$

We find,

$$\mathrm{Adv}^{\boldsymbol{\lambda}-\mathrm{w}(\cdot)\mathrm{AFL\text{-}eCK}}_{\pi,\text{Case 2.1.a}}(\mathcal{A}) = N_P\mathrm{Adv}^{\mathrm{UFCMLA}}_{\mathrm{SIG}}(\mathcal{E}).$$

Case 2.1.b: There exists a session at V^* with X_{V^*} and X_{U^*} (but σ_{V^*} computed by V^* is different from the σ_{V^*} received to U^* with the protocol message)

Game 1. This game is the original game. When the `Test` query is asked, the Game 1 challenger chooses a random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 1$, the real session key is given to $\mathcal{A}$, otherwise a random value chosen from the same session key space is given. This is the original game. Hence,

$$\mathrm{Adv}_{\text{Game 1}}(\mathcal{A}) = \mathrm{Adv}^{\boldsymbol{\lambda}-\mathrm{w}(\cdot)\text{AFL-eCK}}_{\pi,\text{Case 2.1.b}}(\mathcal{A}). \tag{5.40}$$

Game 2. Abort the simulation if there exists two sessions outputting the same ephemeral public keys (Same $X = g^x$ values). Since the ephemeral keys are coming from $\mathbb{Z}_q^*$, the total number of ephemeral keys are q. Total number of session in the simulation is ${N_P}^2{N_s}^2$, because N_P is the number of protocol principals and each protocol principal owns N_s number of sessions. Hence,

$$|\mathrm{Adv}_{\text{Game 1}}(\mathcal{A}) - \mathrm{Adv}_{\text{Game 2}}(\mathcal{A})| \leq \frac{{N_P}^2{N_s}^2}{q}. \tag{5.41}$$

Game 3. Before $\mathcal{A}$ begins, two distinct random principals $U^*, V^* \xleftarrow{\$} \{U_1, ..., U_{N_P}\}$ are chosen as the owner and the peer, and two random numbers $s^*, t^* \xleftarrow{\$} \{1, ...N_s\}$ are chosen, where N_P is the number of protocol principals and N_s is the number of sessions on a principal. The session $\Pi^{s^*}_{U^*,V^*}$ is chosen as the target session and the session $\Pi^{t^*}_{V^*,U^*}$ is chosen as the almost partner session to the target session. If the test session is not the session $\Pi^{s^*}_{U^*,V^*}$, the Game 3 challenger aborts the game. Unless the incorrect choice happens, Game 3 is identical to Game 2. Hence,

$$\mathrm{Adv}_{\text{Game 3}}(\mathcal{A}) = \frac{1}{{N_P}^2 N_s^2}\mathrm{Adv}_{\text{Game 2}}(\mathcal{A}). \tag{5.42}$$

The almost partner session is the session, which communicates with the target session but due to adversaries interaction it does not preserve the partnering conditions.

Game 4. Game 4 challenger randomly chooses a pseudo-ephemeral value $\widetilde{r_{U^*}} \xleftarrow{\$} \mathbb{Z}_q^*$, and computes the ephemeral key $r_{U^*} \xleftarrow{\$} \mathrm{Enc}(p_{U^*}, \widetilde{r_{U^*}})$ in the target session.

We introduce an algorithm $\mathcal{F}$ which is constructed using the adversary $\mathcal{A}$,

against the ϵ-pair-generation indistinguishability challenger (ϵ-PG). $\mathcal{F}$ receives a pair $(r_{U^*}, \widetilde{r_{U^*}})$ such that $\widetilde{r_{U^*}} = \mathrm{Dec}(s_{U^*}, r_{U^*})$. $\mathcal{F}$ uses r_{U^*} as the ephemeral key of U^* and $\widetilde{r_{U^*}}$ as the pseudo-ephemeral key of U^* in the target session.

If a random ephemeral key $r_{U^*} \xleftarrow{\$} \mathbb{Z}_q^*$ is chosen first and the pseudo-ephemeral value $\widetilde{r_{U^*}} \leftarrow \mathrm{Dec}(s_{U^*}, r_{U^*})$ is computed, then the simulation constructed by $\mathcal{F}$ is identical to Game 3. Otherwise if a random pseudo-ephemeral value $\widetilde{r_{U^*}} \xleftarrow{\$} \mathbb{Z}_q^*$ is chosen first and the ephemeral key $r_{U^*} \xleftarrow{\$} \mathrm{Enc}(p_{U^*}, \widetilde{r_{U^*}})$ is computed, then the simulation constructed by $\mathcal{F}$ is identical to Game 4. If $\mathcal{A}$ can distinguish the difference between Game 3 and Game 4, then $\mathcal{F}$ can distinguish whether a message/ciphertext pair (m, c) belongs to the distribution D_1 or D_2 (ϵ-pair-generation indistinguishability challenge). $\mathcal{F}$ can answer all the adversarial queries allowed in this case, because it has all the long-term and ephemeral secret keys of the allowed queries. Hence,

$$|\mathrm{Adv}_{\text{Game 3}}(\mathcal{A}) - \mathrm{Adv}_{\text{Game 4}}(\mathcal{A})| \leq \epsilon. \tag{5.43}$$

Game 5. Game 5 challenger randomly chooses a pseudo-ephemeral value $\widetilde{r'_{U^*}} \xleftarrow{\$} \mathbb{Z}_q^*$, and uses it as the pseudo ephemeral value of U^* in the target session.

We introduce an algorithm $\mathcal{D}$ which is constructed using the adversary $\mathcal{A}$, against the CPLA2 challenger. The algorithm $\mathcal{D}$ uses the public-key of the CPLA2 challenger as the public key of the protocol principal U^* and generates public/secret key pairs for all other protocol principals. $\mathcal{D}$ generates signing/verification key pairs for every protocol principal. $\mathcal{D}$ picks two random strings, $r_0, r_1 \xleftarrow{\$} \{0,1\}^k$ and passes them to the CPLA2 challenger. From the CPLA2 challenger, $\mathcal{D}$ receives a challenge ciphertext C_1 such that $C_1 \xleftarrow{\$} \mathrm{Enc}(p_{U^*}, r_\theta)$ where $r_\theta = r_0$ or $r_\theta = r_1$. The following describes the procedure of answering queries:

- $\mathtt{Send}(U, V, s, m, \mathbf{f})$ query: When $U = U^*$, $V = V^*$ and $s = s^*$, $\mathcal{D}$ takes r_1 as $\widetilde{r'_{U^*}}$, computes $g^{\widetilde{r'_{U^*}}}$ and computes its signature using the signing key sk_{U^*}. Then $\mathcal{D}$ creates the protocol message and sends it to $\mathcal{A}$ with the leakage $\mathbf{f}(s_{U^*})$, where the leakage $\mathbf{f}(s_{U^*})$ is obtained by accessing the leakage oracle of the CPLA2 challenger.

  For all other `Send` queries, $\mathcal{D}$ can execute the protocol normally, because $\mathcal{D}$ has all the public keys and can compute protocol messages accordingly.

Except U^* $\mathcal{D}$ can compute the leakage by its own, and for U^* $\mathcal{D}$ accesses the leakage oracle to obtain the leakage.

- `SessionKeyReveal`(U, V, s) query: $\mathcal{D}$ will abort if `SessionKeyReveal`(U^*, V^*, s^*) or `SessionKeyReveal`(V^*, U^*, t^*) query is asked. $\mathcal{D}$ can easily compute the answers using the corresponding psuedo-ephemeral keys for other `SessionKeyReveal` queries.

- `EphemeralKeyReveal`(U, V, s) query: For the `EphemeralKeyReveal`(U^*, V^*, s^*) query, $\mathcal{D}$ uses C_1 as the answer. For all other `EphemeralKeyReveal` queries $\mathcal{D}$ will answer with the corresponding ephemeral-key which is computed by encrypting a pseudo-ephemeral value with the secret key of the corresponding principal.

- `Corrupt`(U) query: Algorithm $\mathcal{D}$ can answer all the `Corrupt` queries allowed in the freshness condition.

- `Test`(U, s) query: The algorithm $\mathcal{D}$ will abort the game if the adversary issues a `Test` query other than `Test`(U^*, s^*). To compute the answer to the `Test`(U^*, s^*) query, the algorithm $\mathcal{D}$ computes:

  - If U^* is the initiator, computes $ms \leftarrow \mathrm{KDF}(X_{V^*}^{\widetilde{r'_{U^*}}}, \perp, k, \perp)$, $K \leftarrow \mathrm{PRF}(ms, U^*||X_{U^*}||\sigma_{U^*}||V^*||X_{V^*}||\sigma_{V^*})$ where $\widetilde{r'_{U^*}} = r_1$, when U^* is the initiator.
  - If U^* is the responder, computes $ms \leftarrow \mathrm{KDF}(X_{V^*}^{\widetilde{r'_{U^*}}}, \perp, k, \perp)$, $K \leftarrow \mathrm{PRF}(ms, V^*||X_{V^*}||\sigma_{V^*}||U^*||X_{U^*}||\sigma_{U^*})$.

  Then using K answers the `Test` query.

If $\theta = 1$, then r_1 is the decryption of C_1 and the simulation constructed by $\mathcal{D}$ is identical to Game 4 whereas if $\theta = 0$, then r_0 is the decryption of C_1 and the simulation constructed by $\mathcal{D}$ is identical to Game 5. If $\mathcal{A}$ can distinguish the difference between Game 4 and Game 5, then $\mathcal{D}$ can be used against a CPLA2 challenger. Hence,

$$|\mathrm{Adv}_{\text{Game 4}}(\mathcal{A}) - \mathrm{Adv}_{\text{Game 5}}(\mathcal{A})| \leq \mathrm{Adv}_{\mathrm{PKE}}^{\mathrm{CPLA2}}(\mathcal{D}). \tag{5.44}$$

Game 6. Game 6 challenger randomly chooses a pseudo-ephemeral value $\widetilde{r_{V^*}} \xleftarrow{\$} \mathbb{Z}_q^*$, and computes the ephemeral key $r_{V^*} \xleftarrow{\$} \text{Enc}(p_{V^*}, \widetilde{r_{V^*}})$ in the almost partner session.

We introduce an algorithm $\mathcal{F}$ which is constructed using the adversary $\mathcal{A}$, against the ϵ-pair-generation indistinguishability challenger (ϵ-PG). $\mathcal{F}$ receives a pair $(r_{V^*}, \widetilde{r_{V^*}})$ such that $\widetilde{r_{V^*}} = \text{Dec}(s_{V^*}, r_{V^*})$. $\mathcal{F}$ uses r_{V^*} as the ephemeral key of V^* and $\widetilde{r_{V^*}}$ as the pseudo-ephemeral key of V^* in the almost partner session.

If a random ephemeral key $r_{V^*} \xleftarrow{\$} \mathbb{Z}_q^*$ is chosen first and the pseudo-ephemeral value $\widetilde{r_{V^*}} \leftarrow \text{Dec}(s_{V^*}, r_{V^*})$ is computed, then the simulation constructed by $\mathcal{F}$ is identical to Game 5. Otherwise if a random pseudo-ephemeral value $\widetilde{r_{V^*}} \xleftarrow{\$} \mathbb{Z}_q^*$ is chosen first and the ephemeral key $r_{V^*} \xleftarrow{\$} \text{Enc}(p_{V^*}, \widetilde{r_{V^*}})$ is computed, then the simulation constructed by $\mathcal{F}$ is identical to Game 6. If $\mathcal{A}$ can distinguish the difference between Game 5 and Game 6, then $\mathcal{F}$ can distinguish whether a message/ciphertext pair (m, c) belongs to the distribution D_1 or D_2 (ϵ-pair-generation indistinguishability challenge). $\mathcal{F}$ can answer all the adversarial queries allowed in this case, because it has all the long-term and ephemeral secret keys of the allowed queries. Hence,

$$|\text{Adv}_{\text{Game 5}}(\mathcal{A}) - \text{Adv}_{\text{Game 6}}(\mathcal{A})| \leq \epsilon. \tag{5.45}$$

Game 7. Game 7 challenger randomly chooses a pseudo-ephemeral value $\widetilde{r'_{V^*}} \xleftarrow{\$} \mathbb{Z}_q^*$, and uses it as the pseudo ephemeral value of V^* in the almost partner session.

We introduce an algorithm $\mathcal{D}$ which is constructed using the adversary $\mathcal{A}$, against the CPLA2 challenger. The algorithm $\mathcal{D}$ uses the public-key of the CPLA2 challenger as the public key of the protocol principal V^* and generates public/secret key pairs for all other protocol principals. $\mathcal{D}$ generates signing/verification key pairs for every protocol principal. $\mathcal{D}$ picks two random strings, $r'_0, r'_1 \xleftarrow{\$} \{0,1\}^k$ and passes them to the CPLA2 challenger. From the CPLA2 challenger, $\mathcal{D}$ receives a challenge ciphertext C_2 such that $C_2 \xleftarrow{\$} \text{Enc}(p_{V^*}, r'_\theta)$ where $r'_\theta = r'_0$ or $r'_\theta = r'_1$. The following describes the procedure of answering queries:

- $\texttt{Send}(U, V, s, m, \mathbf{f})$ query: When $U = V^*$, $V = U^*$ and $s = t^*$, $\mathcal{D}$ takes r'_1 as $\widetilde{r'_{V^*}}$, computes $g^{\widetilde{r'_{V^*}}}$ and computes its signature using the signing key sk_{V^*}. Then $\mathcal{D}$ creates the protocol message and sends it to $\mathcal{A}$ with the leakage

$\mathbf{f}(s_{V^*})$, where the leakage $\mathbf{f}(s_{V^*})$ is obtained by accessing the leakage oracle of the CPLA2 challenger.

For all other `Send` queries, $\mathcal{D}$ can execute the protocol normally, because $\mathcal{D}$ has all the public keys and can compute protocol messages accordingly. Except V^* $\mathcal{D}$ can compute the leakage by its own, and for V^* $\mathcal{D}$ accesses the leakage oracle to obtain the leakage.

- `SessionKeyReveal`(U, V, s) query: $\mathcal{D}$ will abort if `SessionKeyReveal`(U^*, V^*, s^*) or `SessionKeyReveal`(V^*, U^*, t^*) query is asked. $\mathcal{D}$ can easily compute the answers using the corresponding psuedo-ephemeral keys for other `SessionKeyReveal` queries.

- `EphemeralKeyReveal`(U, V, s) query: For the `EphemeralKeyReveal`(V^*, U^*, t^*) query, $\mathcal{D}$ uses C_2 as the answer. For all other `EphemeralKeyReveal` queries $\mathcal{D}$ will answer with the corresponding ephemeral-key which is computed by encrypting a pseudo-ephemeral value with the secret key of the corresponding principal.

- `Corrupt`(U) query: Algorithm $\mathcal{D}$ can answer all the `Corrupt` queries allowed in the freshness condition.

- `Test`(U, s) query: The algorithm $\mathcal{D}$ will abort the game if the adversary issues a `Test` query other than `Test`(U^*, s^*). To compute the answer to the `Test`(U^*, s^*) query, the algorithm $\mathcal{D}$ computes:

  - If U^* is the initiator, computes $ms \leftarrow \mathrm{KDF}(\widetilde{X^{r'_{V^*}}_{U^*}}, \perp, k, \perp)$, $K \leftarrow \mathrm{PRF}(ms, U^*||X_{U^*}||\sigma_{U^*}||V^*||X_{V^*}||\sigma_{V^*})$ where $\widetilde{r'_{U^*}} = r_1$.
  - If U^* is the responder, computes $ms \leftarrow \mathrm{KDF}(\widetilde{X^{r'_{V^*}}_{U^*}}, \perp, k, \perp)$, $K \leftarrow \mathrm{PRF}(ms, V^*||X_{V^*}||\sigma_{V^*}||U^*||X_{U^*}||\sigma_{U^*})$.

  Then using K answers the `Test` query.

If $\theta = 1$, then r'_1 is the decryption of C_2 and the simulation constructed by $\mathcal{D}$ is identical to Game 6 whereas if $\theta = 0$, then r'_0 is the decryption of C_2 and the simulation constructed by $\mathcal{D}$ is identical to Game 7. If $\mathcal{A}$ can distinguish the difference between Game 6 and Game 7, then $\mathcal{D}$ can be used against a CPLA2 challenger. Hence,

$$|\mathrm{Adv}_{\text{Game } 6}(\mathcal{A}) - \mathrm{Adv}_{\text{Game } 7}(\mathcal{A})| \leq \mathrm{Adv}^{\mathrm{CPLA2}}_{\mathrm{PKE}}(\mathcal{D}). \tag{5.46}$$

Game 8. Game 8 challenger randomly chooses $z \xleftarrow{\$} \mathbb{Z}_q^*$ and computes the session key of the target session, using the KDF and the PRF as $ms \leftarrow \text{KDF}(g^z, \perp, k, \perp)$ and $K \leftarrow \text{PRF}(ms, U^*||X_{U^*}||\sigma_{U^*}||V^*||X_{V^*}||\sigma_{V^*})$, when U^* is the initiator, or $K \leftarrow \text{PRF}(ms, V^*||X_{V^*}||\sigma_{V^*}||U^*||X_{U^*}||\sigma_{U^*})$, when U^* is the responder.

We construct an algorithm $\mathcal{C}$ against the DDH challenge, using the adversary $\mathcal{A}$. The DDH challenger sends values $(X = g^x, Y = g^y, Z = g^z)$ such that either $z = xy$ or $z \xleftarrow{\$} \mathbb{Z}_q^*$, as the inputs to the algorithm $\mathcal{C}$. $\mathcal{C}$ uses the value X as the ephemeral public key of U^* and Y as the ephemeral public key of V^* in the test session, and computes the session key using Z as the input to the KDF in the session key derivation process. In this game, for the almost partner session, the value Z is used as the Diffie-Hellman shared secret.

If $\mathcal{C}$'s input is a Diffie-Hellman triple, the simulation constructed by $\mathcal{C}$ is identical to Game 7, otherwise it is identical to Game 8. If $\mathcal{A}$ can distinguish whether $g^z = g^{xy}$ or not, then $\mathcal{C}$ can answer the DDH challenge. $\mathcal{C}$ can answer all the adversarial queries allowed in this case, because it has all the long-term and ephemeral secret keys of the allowed queries. Hence,

$$|\text{Adv}_{\text{Game } 7}(\mathcal{A}) - \text{Adv}_{\text{Game } 8}(\mathcal{A})| \leq \text{Adv}_{q,g}^{\text{DDH}}(\mathcal{C}). \tag{5.47}$$

Game 9. The Game 9 challenger randomly chooses $ms \xleftarrow{\$} \{0,1\}^k$ and computes the session key of the target session, using the PRF as $K \leftarrow \text{PRF}(ms, U^*||X_{U^*}||\sigma_{U^*}||V^*||X_{V^*}||\sigma_{V^*})$, when U^* is the initiator or $K \leftarrow \text{PRF}(ms, V^*||X_{V^*}||\sigma_{V^*}||U^*||X_{U^*}||\sigma_{U^*})$, when U^* is the responder.

We construct an algorithm $\mathcal{B}$ against a KDF challenger, using the adversary $\mathcal{A}$. The KDF challenger sends a ms value which is either generated using the KDF or randomly chosen. $\mathcal{B}$ uses the received ms value to compute the session key of the target session using the PRF. In this game, for the almost partner session, the value ms is used as the shared value derived using the KDF.

If ms is computed using the KDF, simulation constructed by $\mathcal{B}$ is identical to Game 8, otherwise it is identical to Game 9. If $\mathcal{A}$ can distinguish the difference between Game 8 and Game 9, then $\mathcal{A}$ can be used as a subroutine of an algorithm $\mathcal{B}$, which is used to distinguish whether the ms value is computed using KDF or randomly chosen. $\mathcal{B}$ can answer all the adversarial queries allowed in this case, because it has all the long-term and ephemeral secret keys of the allowed queries. Hence,

$$|\mathrm{Adv}_{\text{Game }8}(\mathcal{A}) - \mathrm{Adv}_{\text{Game }9}(\mathcal{A})| \leq \mathrm{Adv}_{\mathrm{KDF}}(\mathcal{B}). \tag{5.48}$$

Game 10. The Game 10 challenger randomly chooses $K \xleftarrow{\$} \{0,1\}^k$ as session key of the target session.

We construct an algorithm $\mathcal{J}$ against an Oracle$^{\mathrm{PRF}}$, using the adversary $\mathcal{A}$. The Oracle$^{\mathrm{PRF}}$ sends a K value which is either generated using the PRF with a hidden key, or a random function. $\mathcal{J}$ uses the received K as the session key of the target session.

If K is generated using the PRF with a hidden key, simulation constructed by $\mathcal{J}$ is identical to Game 9, otherwise it is identical to Game 10. If $\mathcal{A}$ can distinguish the difference between Game 9 and Game 10, then $\mathcal{A}$ can be used as a subroutine of an algorithm $\mathcal{J}$, which is used to distinguish whether the Oracle$^{\mathrm{PRF}}$ is real or a random function. $\mathcal{J}$ can answer all the adversarial queries allowed in this case, because it has all the long-term and ephemeral secret keys of the allowed queries. Hence,

$$|\mathrm{Adv}_{\text{Game }9}(\mathcal{A}) - \mathrm{Adv}_{\text{Game }10}(\mathcal{A})| \leq \mathrm{Adv}_{\mathrm{PRF}}(\mathcal{J}). \tag{5.49}$$

Semantic security of the session key in Game 10. Since the session key K of $\Pi^{s^*}_{U^*,V^*}$ is chosen randomly and independently from all other values, $\mathcal{A}$ does not have any advantage in Game 10. Hence,

$$\mathrm{Adv}_{\text{Game }10}(\mathcal{A}) = 0. \tag{5.50}$$

We find,

$$\mathrm{Adv}^{\boldsymbol{\lambda}-\mathrm{w}(\cdot)\mathrm{AFL\text{-}eCK}}_{\pi,\text{Case 2.1.b}}(\mathcal{A}) \leq N_P^2 N_s^2 \Big[\big(\mathrm{Adv}^{\mathrm{DDH}}_{q,g}(\mathcal{C}) + \mathrm{Adv}_{\mathrm{KDF}}(\mathcal{B}) + \mathrm{Adv}_{\mathrm{PRF}}(\mathcal{J}) + 2\mathrm{Adv}^{\mathrm{CPLA2}}_{\mathrm{PKE}}(\mathcal{D}) + 2\epsilon \big) + \frac{1}{q} \Big].$$

Case 2.1.c: There exists a session at V^* with X_{V^*} and $X'_{U^*} \neq X_{U^*}$

Game 1. This game is the original game. When the `Test` query is asked, the Game 1 challenger chooses a random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 1$, the real session key is given to $\mathcal{A}$, otherwise a random value chosen from the same session key space

is given. This is the original game. Hence,

$$\mathrm{Adv}_{\text{Game 1}}(\mathcal{A}) = \mathrm{Adv}_{\pi}^{\boldsymbol{\lambda}-\mathrm{w}(\cdot)\text{AFL-eCK}}(\mathcal{A}). \tag{5.51}$$

Game 2. Abort the simulation if there exists two sessions outputting the same ephemeral public keys (Same $X = g^x$ values). Since the ephemeral keys are coming from $\mathbb{Z}_q^*$, the total number of ephemeral keys are q. Total number of session in the simulation is ${N_P}^2{N_s}^2$, because N_P is the number of protocol principals and each protocol principal owns N_s number of sessions. Hence,

$$|\mathrm{Adv}_{\text{Game 1}}(\mathcal{A}) - \mathrm{Adv}_{\text{Game 2}}(\mathcal{A})| \leq \frac{{N_P}^2{N_s}^2}{q}. \tag{5.52}$$

Game 3. Before $\mathcal{A}$ begins, two distinct random principals $U^*, V^* \xleftarrow{\$} \{U_1, ..., U_{N_P}\}$ are chosen as the owner and the peer, and two random numbers $s^*, t^* \xleftarrow{\$} \{1, ...N_s\}$ are chosen, where N_P is the number of protocol principals and N_s is the number of sessions on a principal. The session $\Pi^{s^*}_{U^*,V^*}$ is chosen as the target session and the session $\Pi^{t^*}_{V^*,U^*}$ is chosen as the almost partner session to the target session. If the test session is not the session $\Pi^{s^*}_{U^*,V^*}$, the Game 3 challenger aborts the game. Unless the incorrect choice happens, Game 3 is identical to Game 2. Hence,

$$\mathrm{Adv}_{\text{Game 3}}(\mathcal{A}) = \frac{1}{{N_P}^2 N_s^2}\mathrm{Adv}_{\text{Game 2}}(\mathcal{A}). \tag{5.53}$$

Game 4. Game 4 challenger randomly chooses a pseudo-ephemeral value $\widetilde{r_{U^*}} \xleftarrow{\$} \mathbb{Z}_q^*$, and computes the ephemeral key $r_{U^*} \xleftarrow{\$} \mathrm{Enc}(p_{U^*}, \widetilde{r_{U^*}})$ in the target session.

We introduce an algorithm $\mathcal{F}$ which is constructed using the adversary $\mathcal{A}$, against the ϵ-pair-generation indistinguishability challenger (ϵ-PG). $\mathcal{F}$ receives a pair $(r_{U^*}, \widetilde{r_{U^*}})$ such that $\widetilde{r_{U^*}} = \mathrm{Dec}(s_{U^*}, r_{U^*})$. $\mathcal{F}$ uses r_{U^*} as the ephemeral key of U^* and $\widetilde{r_{U^*}}$ as the pseudo-ephemeral key of U^* in the target session.

If a random ephemeral key $r_{U^*} \xleftarrow{\$} \mathbb{Z}_q^*$ is chosen first and the pseudo-ephemeral value $\widetilde{r_{U^*}} \leftarrow \mathrm{Dec}(s_{U^*}, r_{U^*})$ is computed, then the simulation constructed by $\mathcal{F}$ is identical to Game 3. Otherwise if a random pseudo-ephemeral value $\widetilde{r_{U^*}} \xleftarrow{\$} \mathbb{Z}_q^*$ is chosen first and the ephemeral key $r_{U^*} \xleftarrow{\$} \mathrm{Enc}(p_{U^*}, \widetilde{r_{U^*}})$ is computed, then the simulation constructed by $\mathcal{F}$ is identical to Game 4. If $\mathcal{A}$ can distinguish the difference between Game 3 and Game 4, then $\mathcal{F}$ can distinguish whether

a message/ciphertext pair (m, c) belongs to the distribution D_1 or D_2 (ϵ-pair-generation indistinguishability challenge). $\mathcal{F}$ can answer all the adversarial queries allowed in this case, because it has all the long-term and ephemeral secret keys of the allowed queries. Hence,

$$|\mathrm{Adv}_{\mathrm{Game}\ 3}(\mathcal{A}) - \mathrm{Adv}_{\mathrm{Game}\ 4}(\mathcal{A})| \leq \epsilon. \tag{5.54}$$

Game 5. Game 5 challenger randomly chooses a pseudo-ephemeral value $\widetilde{r'_{U^*}} \xleftarrow{\$} \mathbb{Z}_q^*$, and uses it as the pseudo ephemeral value of U^* in the target session.

We introduce an algorithm $\mathcal{D}$ which is constructed using the adversary $\mathcal{A}$, against the CPLA2 challenger. The algorithm $\mathcal{D}$ uses the public-key of the CPLA2 challenger as the public key of the protocol principal U^* and generates public/secret key pairs for all other protocol principals. $\mathcal{D}$ generates signing/verification key pairs for every protocol principal. $\mathcal{D}$ picks two random strings, $r_0, r_1 \xleftarrow{\$} \{0,1\}^k$ and passes them to the CPLA2 challenger. From the CPLA2 challenger, $\mathcal{D}$ receives a challenge ciphertext C_1 such that $C_1 \xleftarrow{\$} \mathrm{Enc}(p_{U^*}, r_\theta)$ where $r_\theta = r_0$ or $r_\theta = r_1$. The following describes the procedure of answering queries:

- `Send`$(U, V, s, m, \mathbf{f})$ query: When $U = U^*$, $V = V^*$ and $s = s^*$, $\mathcal{D}$ takes r_1 as $\widetilde{r'_{U^*}}$, computes $g^{\widetilde{r'_{U^*}}}$ and computes its signature using the signing key sk_{U^*}. Then $\mathcal{D}$ creates the protocol message and sends it to $\mathcal{A}$ with the leakage $\mathbf{f}(s_{U^*})$, where the leakage $\mathbf{f}(s_{U^*})$ is obtained by accessing the leakage oracle of the CPLA2 challenger.

  For all other `Send` queries, $\mathcal{D}$ can execute the protocol normally, because $\mathcal{D}$ has all the public keys and can compute protocol messages accordingly. Except U^* $\mathcal{D}$ can compute the leakage by its own, and for U^* $\mathcal{D}$ accesses the leakage oracle to obtain the leakage.

- `SessionKeyReveal`(U, V, s) query: $\mathcal{D}$ will abort if `SessionKeyReveal`(U^*, V^*, s^*) or `SessionKeyReveal`(V^*, U^*, t^*) query is asked. $\mathcal{D}$ can easily compute the answers using the corresponding psuedo-ephemeral keys for other `SessionKeyReveal` queries.

- `EphemeralKeyReveal`(U, V, s) query: For the `EphemeralKeyReveal`(U^*, V^*, s^*) query, $\mathcal{D}$ uses C_1 as the answer. For all other `EphemeralKeyReveal`

queries $\mathcal{D}$ will answer with the corresponding ephemeral-key which is computed by encrypting a pseudo-ephemeral value with the secret key of the corresponding principal.

- $\texttt{Corrupt}(U)$ query: Algorithm $\mathcal{D}$ can answer all the `Corrupt` queries allowed in the freshness condition.

- $\texttt{Test}(U, s)$ query: The algorithm $\mathcal{D}$ will abort the game if the adversary issues a `Test` query other than $\texttt{Test}(U^*, s^*)$. To compute the answer to the $\texttt{Test}(U^*, s^*)$ query, the algorithm $\mathcal{D}$ computes:

  – If U^* is the initiator, $ms \leftarrow \text{KDF}(X_{V^*}^{\widetilde{r'_{U^*}}}, \perp, k, \perp)$, $K \leftarrow \text{PRF}(ms, U^* ||X_{U^*}||\sigma_{U^*}||V^*||X_{V^*}||\sigma_{V^*})$ where $\widetilde{r'_{U^*}} = r_1$.

  – If U^* is the responder computes $ms \leftarrow \text{KDF}(X_{V^*}^{\widetilde{r'_{U^*}}}, \perp, k, \perp)$, $K \leftarrow \text{PRF}(ms, V^*||X_{V^*}||\sigma_{V^*}||U^*||X_{U^*}||\sigma_{U^*})$.

  Then using K answers the `Test` query.

If $\theta = 1$, then r_1 is the decryption of C_1 and the simulation constructed by $\mathcal{D}$ is identical to Game 4 whereas if $\theta = 0$, then r_0 is the decryption of C_1 and the simulation constructed by $\mathcal{D}$ is identical to Game 5. If $\mathcal{A}$ can distinguish the difference between Game 4 and Game 5, then $\mathcal{D}$ can be used against a CPLA2 challenger. Hence,

$$|\text{Adv}_{\text{Game 4}}(\mathcal{A}) - \text{Adv}_{\text{Game 5}}(\mathcal{A})| \leq \text{Adv}_{\text{PKE}}^{\text{CPLA2}}(\mathcal{D}). \tag{5.55}$$

Game 6. Game 6 challenger randomly chooses a pseudo-ephemeral value $\widetilde{r_{V^*}} \xleftarrow{\$} \mathbb{Z}_q^*$, and computes the ephemeral key $r_{V^*} \xleftarrow{\$} \text{Enc}(p_{V^*}, \widetilde{r_{V^*}})$ in the almost partner session.

We introduce an algorithm $\mathcal{F}$ which is constructed using the adversary $\mathcal{A}$, against the ϵ-pair-generation indistinguishability challenger (ϵ-PG). $\mathcal{F}$ receives a pair $(r_{V^*}, \widetilde{r_{V^*}})$ such that $\widetilde{r_{V^*}} = \text{Dec}(s_{V^*}, r_{V^*})$. $\mathcal{F}$ uses r_{V^*} as the ephemeral key of V^* and $\widetilde{r_{V^*}}$ as the pseudo-ephemeral key of V^* in the almost partner session.

If a random ephemeral key $r_{V^*} \xleftarrow{\$} \mathbb{Z}_q^*$ is chosen first and the pseudo-ephemeral value $\widetilde{r_{V^*}} \leftarrow \text{Dec}(s_{V^*}, r_{V^*})$ is computed, then the simulation constructed by $\mathcal{F}$ is identical to Game 5. Otherwise if a random pseudo-ephemeral value $\widetilde{r_{V^*}} \xleftarrow{\$} \mathbb{Z}_q^*$ is chosen first and the ephemeral key $r_{V^*} \xleftarrow{\$} \text{Enc}(p_{V^*}, \widetilde{r_{V^*}})$ is computed, then

the simulation constructed by $\mathcal{F}$ is identical to Game 6. If $\mathcal{A}$ can distinguish the difference between Game 5 and Game 6, then $\mathcal{F}$ can distinguish whether a message/ciphertext pair (m, c) belongs to the distribution D_1 or D_2 (ϵ-pair-generation indistinguishability challenge). $\mathcal{F}$ can answer all the adversarial queries allowed in this case, because it has all the long-term and ephemeral secret keys of the allowed queries. Hence,

$$|\mathrm{Adv}_{\text{Game } 5}(\mathcal{A}) - \mathrm{Adv}_{\text{Game } 6}(\mathcal{A})| \leq \epsilon. \tag{5.56}$$

Game 7. Game 7 challenger randomly chooses a pseudo-ephemeral value $\widetilde{r'_{V^*}} \xleftarrow{\$} \mathbb{Z}_q^*$, and uses it as the pseudo ephemeral value of V^* in the almost partner session.

We introduce an algorithm $\mathcal{D}$ which is constructed using the adversary $\mathcal{A}$, against the CPLA2 challenger. The algorithm $\mathcal{D}$ uses the public-key of the CPLA2 challenger as the public key of the protocol principal V^* and generates public/secret key pairs for all other protocol principals. $\mathcal{D}$ generates signing/verification key pairs for every protocol principal. $\mathcal{D}$ picks two random strings, $r'_0, r'_1 \xleftarrow{\$} \{0,1\}^k$ and passes them to the CPLA2 challenger. From the CPLA2 challenger, $\mathcal{D}$ receives a challenge ciphertext C_2 such that $C_2 \xleftarrow{\$} \mathrm{Enc}(p_{V^*}, r'_\theta)$ where $r'_\theta = r'_0$ or $r'_\theta = r'_1$. The following describes the procedure of answering queries:

- `Send`$(U, V, s, m, \mathbf{f})$ query: When $U = V^*$, $V = U^*$ and $s = t^*$, $\mathcal{D}$ takes r'_1 as $\widetilde{r'_{V^*}}$, computes $g^{\widetilde{r'_{V^*}}}$ and computes its signature using the signing key sk_{V^*}. Then $\mathcal{D}$ creates the protocol message and sends it to $\mathcal{A}$ with the leakage $\mathbf{f}(s_{V^*})$, where the leakage $\mathbf{f}(s_{V^*})$ is obtained by accessing the leakage oracle of the CPLA2 challenger.

  For all other `Send` queries, $\mathcal{D}$ can execute the protocol normally, because $\mathcal{D}$ has all the public keys and can compute protocol messages accordingly. Except V^* $\mathcal{D}$ can compute the leakage by its own, and for V^* $\mathcal{D}$ accesses the leakage oracle to obtain the leakage.

- `SessionKeyReveal`(U, V, s) query: $\mathcal{D}$ will abort if `SessionKeyReveal`(U^*, V^*, s^*) or `SessionKeyReveal`(V^*, U^*, t^*) query is asked. $\mathcal{D}$ can easily compute the answers using the corresponding psuedo-ephemeral keys for other `SessionKeyReveal` queries.

- `EphemeralKeyReveal`(U, V, s) query: For the `EphemeralKeyReveal`(V^*, U^*, t^*) query, $\mathcal{D}$ uses C_2 as the answer. For all other `EphemeralKeyReveal` queries $\mathcal{D}$ will answer with the corresponding ephemeral-key which is computed by encrypting a pseudo-ephemeral value with the secret key of the corresponding principal.

- `Corrupt`(U) query: Algorithm $\mathcal{D}$ can answer all the `Corrupt` queries allowed in the freshness condition.

- `Test`(U, s) query: The algorithm $\mathcal{D}$ will abort the game if the adversary issues a `Test` query other than `Test`(U^*, s^*). To compute the answer to the `Test`(U^*, s^*) query, the algorithm $\mathcal{D}$ computes:

  - If U^* is the initiator, $ms \leftarrow \text{KDF}(\widetilde{X_{U^*}^{r'_{V^*}}}, \perp, k, \perp)$, $K \leftarrow \text{PRF}(ms, U^* ||X_{U^*}||\sigma_{U^*}||V^*||X_{V^*}||\sigma_{V^*})$ where $\widetilde{r'_{U^*}} = r_1$.
  - If U^* is the responder computes $ms \leftarrow \text{KDF}(\widetilde{X_{U^*}^{r'_{V^*}}}, \perp, k, \perp)$, $K \leftarrow \text{PRF}(ms, V^*||X_{V^*}||\sigma_{V^*}||U^*||X_{U^*}||\sigma_{U^*})$.

  Then using K answers the `Test` query.

If $\theta = 1$, then r'_1 is the decryption of C_2 and the simulation constructed by $\mathcal{D}$ is identical to Game 6 whereas if $\theta = 0$, then r'_0 is the decryption of C_2 and the simulation constructed by $\mathcal{D}$ is identical to Game 7. If $\mathcal{A}$ can distinguish the difference between Game 6 and Game 7, then $\mathcal{D}$ can be used against a CPLA2 challenger. Hence,

$$|\text{Adv}_{\text{Game }6}(\mathcal{A}) - \text{Adv}_{\text{Game }7}(\mathcal{A})| \leq \text{Adv}_{\text{PKE}}^{\text{CPLA2}}(\mathcal{D}). \tag{5.57}$$

Game 8. Game 8 challenger randomly chooses $ms \xleftarrow{\$} \{0,1\}^k$ and computes the session key of the target session, using the PRF as $K \leftarrow \text{PRF}(ms, U^*||X_{U^*}||\sigma_{U^*}||V^*||X_{V^*}||\sigma_{V^*})$, when U^* is the initiator or $K \leftarrow \text{PRF}(ms, V^*||X_{V^*}||\sigma_{V^*}||U^*||X_{U^*}||\sigma_{U^*})$, when U^* is the responder.

We construct an algorithm $\mathcal{R}$ against the ODH challenge, using the adversary $\mathcal{A}$. The ODH challenge being for group $\mathbb{G}$ with prime order q, generator g and function $\text{KDF}(\cdot, \perp, k, \cdot)$. The ODH challenger sends values $(X = g^x, Y = g^y, Z)$ such that either $Z \leftarrow \text{KDF}(g^{xy}, \perp, k, \perp)$ or $Z \xleftarrow{\$} \{0,1\}^k$, as the inputs to the algorithm $\mathcal{R}$. $\mathcal{R}$ uses the value X as the ephemeral public key of U^* and Y as the

ephemeral public key of V^* in the test session, and computes the session key using Z as the ms value, which is the input to the PRF, in the session key derivation process. In this game, for the almost partner session (a session at V^* with X_{V^*} and $X'_{U^*} \neq X_{U^*}$), the ODH oracle $\mathcal{O}^{\text{ODH}}$ is used to compute the ms value. For all the other honest sessions, simulator knows the ephemeral keys and can compute the shared Diffie-Hellman value and compute the session key normally.

If $\mathcal{R}$'s input $Z = \text{KDF}(g^{xy}, \perp, k, \perp)$, the simulation constructed by $\mathcal{R}$ is identical to Game 7, otherwise it is identical to Game 8. Hence,

$$|\text{Adv}_{\text{Game }7}(\mathcal{A}) - \text{Adv}_{\text{Game }8}(\mathcal{A})| \leq \text{Adv}^{\text{ODH}}_{\text{KDF}(\cdot,\perp,k,\perp),q,g}(\mathcal{R}). \tag{5.58}$$

Game 9. The Game 9 challenger randomly chooses $K \xleftarrow{\$} \{0,1\}^k$ as session key of the target session.

We construct an algorithm $\mathcal{J}$ against an Oracle$^{\text{PRF}}$, using the adversary $\mathcal{A}$. The Oracle$^{\text{PRF}}$ sends a K value which is either generated using the PRF with a hidden key, or a random function. $\mathcal{J}$ uses the received K as the session key of the target session.

If K is generated using the PRF with a hidden key, simulation constructed by $\mathcal{J}$ is identical to Game 8, otherwise it is identical to Game 9. If $\mathcal{A}$ can distinguish the difference between Game 8 and Game 9, then $\mathcal{A}$ can be used as a subroutine of an algorithm $\mathcal{J}$, which is used to distinguish whether the Oracle$^{\text{PRF}}$ is real or a random function. $\mathcal{J}$ can answer all the adversarial queries allowed in this case, because it has all the long-term and ephemeral secret keys of the allowed queries. Hence,

$$|\text{Adv}_{\text{Game }8}(\mathcal{A}) - \text{Adv}_{\text{Game }9}(\mathcal{A})| \leq \text{Adv}_{\text{PRF}}(\mathcal{J}). \tag{5.59}$$

Semantic security of the session key in Game 9. Since the session key K of $\Pi^{s^*}_{U^*,V^*}$ is chosen randomly and independently from all other values, $\mathcal{A}$ does not have any advantage in Game 9. Hence,

$$\text{Adv}_{\text{Game }9}(\mathcal{A}) = 0. \tag{5.60}$$

We find,

$$\mathrm{Adv}^{\boldsymbol{\lambda}-\mathrm{w}(\cdot)\mathrm{AFL\text{-}eCK}}_{\pi,\text{Case 2.1.c}}(\mathcal{A}) \leq N_P^2 N_s^2 \Big[\big(\mathrm{Adv}^{\mathrm{ODH}}_{q,g}(\mathcal{R}) + \mathrm{Adv}_{\mathrm{PRF}}(\mathcal{J}) + 2\mathrm{Adv}^{\mathrm{CPLA2}}_{\mathrm{PKE}}(\mathcal{D}) + 2\epsilon \big) + \frac{1}{q} \Big].$$

Therefore, in Case 2.1,

$$\mathrm{Adv}^{\boldsymbol{\lambda}-\mathrm{w}(\cdot)\mathrm{AFL\text{-}eCK}}_{\pi,\text{Case 2.1}}(\mathcal{A}) \leq \max \Big[N_P^2 N_s^2 \big[\big(\mathrm{Adv}^{\mathrm{DDH}}_{q,g}(\mathcal{C}) + \mathrm{Adv}_{\mathrm{KDF}}(\mathcal{B}) + \mathrm{Adv}_{\mathrm{PRF}}(\mathcal{J}) + 2\mathrm{Adv}^{\mathrm{CPLA2}}_{\mathrm{PKE}}(\mathcal{D}) + 2\epsilon \big) + \frac{1}{q} \big], N_P^2 N_s^2 \big[\big(\mathrm{Adv}^{\mathrm{ODH}}_{q,g}(\mathcal{R}) + \mathrm{Adv}_{\mathrm{PRF}}(\mathcal{J}) + 2\mathrm{Adv}^{\mathrm{CPLA2}}_{\mathrm{PKE}}(\mathcal{D}) + 2\epsilon \big) + \frac{1}{q} \big], N_P \mathrm{Adv}^{\mathrm{UFCMLA}}_{\mathrm{SIG}}(\mathcal{E}) \Big].$$

Case 2.2: Test session is at the initiator Let U^* be the initiator and V^* be the responder. Let X_{U^*} be the ephemeral public key of U^*, and X_{V^*} be the ephemeral public key of V^*, in the target session. In this case there are three sub cases, which address the three different situations occur when the challenger interacts with the adversary, which are same as to the Case 2.1.

- (a) There is no session at V^* with X_{V^*}.
- (b) There exists a session at V^* with X_{U^*} and X_{V^*} (but σ_{U^*} computed by U^* is different from the σ_{U^*} received to V^* with the protocol message, in the target session).
- (c) There exists a session at V^* with X_{V^*} and $X'_{U^*} \neq X_{U^*}$.

This is almost same as the Case 2.1. The difference is that in this case the initiator is the owner of the test session. Same as to the Case 2.1, here we obtain,

$$\mathrm{Adv}^{\boldsymbol{\lambda}-\mathrm{w}(\cdot)\mathrm{AFL\text{-}eCK}}_{\pi,\text{Case 2.2}}(\mathcal{A}) \leq \max \Big[N_P^2 N_s^2 \big[\big(\mathrm{Adv}^{\mathrm{DDH}}_{q,g}(\mathcal{C}) + \mathrm{Adv}_{\mathrm{KDF}}(\mathcal{B}) + \mathrm{Adv}_{\mathrm{PRF}}(\mathcal{J}) + 2\mathrm{Adv}^{\mathrm{CPLA2}}_{\mathrm{PKE}}(\mathcal{D}) + 2\epsilon \big) + \frac{1}{q} \big], N_P^2 N_s^2 \big[\big(\mathrm{Adv}^{\mathrm{ODH}}_{q,g}(\mathcal{R}) + \mathrm{Adv}_{\mathrm{PRF}}(\mathcal{J}) + 2\mathrm{Adv}^{\mathrm{CPLA2}}_{\mathrm{PKE}}(\mathcal{D}) + 2\epsilon \big) + \frac{1}{q} \big], N_P \mathrm{Adv}^{\mathrm{UFCMLA}}_{\mathrm{SIG}}(\mathcal{E}) \Big].$$

Combining Case 1 and Case 2

According to the analysis we can obtain,

$$\mathrm{Adv}_{\pi}^{\boldsymbol{\lambda}-\mathrm{w}(\cdot)\mathrm{AFL\text{-}eCK}}(\mathcal{A}) \leq \max\Big[N_P^2N_s^2\big[\big(\mathrm{Adv}_{q,g}^{\mathrm{DDH}}(\mathcal{C}) + \mathrm{Adv}_{\mathrm{KDF}}(\mathcal{B}) + \mathrm{Adv}_{\mathrm{PRF}}(\mathcal{J})\big) + \frac{1}{q}\big],$$
$$N_P^2N_s^2\big[\big(\mathrm{Adv}_{q,g}^{\mathrm{DDH}}(\mathcal{C}) + \mathrm{Adv}_{\mathrm{KDF}}(\mathcal{B}) + \mathrm{Adv}_{\mathrm{PRF}}(\mathcal{J}) + 2\mathrm{Adv}_{\mathrm{PKE}}^{\mathrm{CPLA2}}(\mathcal{D}) + 2\epsilon\big) + \frac{1}{q}\big],$$
$$N_P^2N_s^2\big[\big(\mathrm{Adv}_{q,g}^{\mathrm{ODH}}(\mathcal{R}) + \mathrm{Adv}_{\mathrm{PRF}}(\mathcal{J}) + 2\mathrm{Adv}_{\mathrm{PKE}}^{\mathrm{CPLA2}}(\mathcal{D}) + 2\epsilon\big) + \frac{1}{q}\big], N_P\mathrm{Adv}_{\mathrm{SIG}}^{\mathrm{UFCMLA}}(\mathcal{E})\Big].$$

□

5.3.6 Leakage Tolerance of the w(·)AFL-eCK-secure Protocol π: wBAFL-eCK-Secure Instantiation

In the presented protocol, a principal uses a decryption function of a CPLA2-secure $\epsilon -$ PG-IND-public-key cryptosystem to compute the NAXOS value in a leakage-resilient manner and sets the Diffie-Hellman exponent as the decrypted message. Then, the principal uses a UFCMLA-secure signature scheme to authenticate the message. Therefore, the leakage tolerance from the secret key used to compute the NAXOS value is exactly same as the leakage tolerance of the underlying CPLA2-secure public key encryption scheme, and the leakage tolerance from the secret key used to compute the signature is exactly same as the leakage tolerance of the underlying UFCMLA-secure signature scheme.

As noted in Section 5.3.2, Halevi and Lin [HL11] constructed a generic CPLA2-secure public-key cryptosystem which is secure against bounded leakage and also satisfies pair generation indistinguishability. It can be instantiated with the DDH-based leakage-resilient public-key encryption scheme of Naor and Segev [NS09] with decryption cost of 4 exponentiations, and for a key length k the leakage is bounded by $(1 - o(1))k$. Katz and Vaikuntanathan [KV09] constructed an UFCMLA-secure signature scheme in the bounded leakage model, where a signature can be generated with cost of 2 exponentiations, and verified with cost of 4 exponentiations (with a simple NIZK proof). The signature scheme of Katz and Vaikuntanathan contains signing and verification operations based on NIZK protocols. For a key length k, the signature scheme tolerates leakage of $(1 - k^t) \cdot k$, for any constant $t < 1$. Hence, this protocol can be instantiated with the above mentioned leakage-resilient signature scheme and the public-

key encryption scheme, and achieve leakage tolerance according to the leakage parameters specified in the above mentioned cryptographic constructions.

5.4 Concrete Construction of CAFL-eCK-secure Key Exchange Protocol

In section 5.3 we presented a generic construction for a protocol which is proven secure in the w(·)AFL-eCK security model. However, when it comes to a concrete construction, the presented generic protocol can only be instantiated in a way that is secure in the *bounded* version of the security model and the model we considered is slightly weaker than the desired (·)AFL-eCK model. Up to now there are no suitable cryptographic primitives which can be used to instantiate the generic protocol in the continuous leakage variant of the security model, as well as in the desired (·)AFL-eCK model. Now our aim is to present a concrete protocol construction which can be proven secure in the continuous leakage instantiation of the generic (·)AFL-eCK, namely CAFL-eCK model. Moreover, this construction does not require a weaker variant of the model for the security proof. Thus, we introduce the first concrete construction of a *continuous* and *after-the-fact* leakage-resilient key exchange protocol.

Moving to the leakage-resilient setting of the eCK-style secure key exchange requires rethinking the NAXOS trick. We have presented a generic construction of a weak after-the-fact leakage eCK (w(·)AFL-eCK)-secure key exchange protocol in Section 5.3, which uses a leakage-resilient NAXOS trick. The leakage-resilient NAXOS trick is obtained using a decryption function of an after-the-fact leakage-resilient public key encryption scheme. A concrete construction of a wBAFL-eCK-secure protocol is possible since there exists a bounded after-the-fact leakage-resilient public key encryption scheme which can be used to obtain the required leakage-resilient NAXOS trick, but it is not currently possible to construct a CAFL-eCK-secure protocol since there is currently no continuous after-the-fact leakage-resilient public-key encryption scheme available. Therefore, an attempt to construct a CAFL-eCK-secure key exchange protocol using the leakage-resilient NAXOS approach is not possible at this stage.

In section 3.3 we presented a eCK-secure protocol construction which does not use the NAXOS trick, namely the protocol P1. The protocol P1 is based on Diffie-Hellman key exchange, which requires exponentiation computations. Moving to

the leakage-resilient setting requires rethinking the exponentiation computation in a leakage-resilient manner. Since there exist leakage-resilient encoding schemes and leakage-resilient refreshing protocols for them (Definition 2.4.1 and 2.4.4), our aim is to compute the required exponentiations in a leakage-resilient manner using the available leakage-resilient storage and refreshing schemes.

5.4.1 Leakage-Resilient Construction of Protocol P1: Protocol P2

Protocol P1 is an eCK-secure key exchange protocol (Theorem 3.3.1). The eCK model considers an environment where partial information leakage does not take place. Following the concept that only computation leaks information, we now assume that the leakage of long-term secret keys happens when computations are performed using them. Then, instead of the *non-leakage* eCK model which we used for the security proof of protocol P1, we consider the CAFL-eCK model which additionally allows the adversary to obtain continuous leakage of long-term secret keys.

Our idea is to perform the computations which use long-term secret keys (exponentiation operations) in such a way that the resulting leakage from the long-term secrets should not leak sufficient information to reveal them to the adversary. To overcome that challenge we use a leakage-resilient storage scheme and a leakage-resilient refreshing protocol, and modify the architecture of the protocol P1, in such a way that the secret keys s are encoded into two portions s_L, s_R, Exponentiations are computed using two portions s_L, s_R instead of directly using s, and the two portions s_L, s_R are being refreshed continuously. Thus, we add leakage resiliency to the eCK-secure protocol P1 and construct protocol P2 such that it is leakage-resilient and eCK-secure.

Obtaining Leakage Resiliency by Encoding Secrets

In this setting we encode a secret s using an Encode function of a leakage-resilient storage scheme $\Lambda = (\text{Encode}, \text{Decode})$. So the secret s is encoded as $(s_L, s_R) \leftarrow \text{Encode}(s)$. As mentioned in the Definition 2.4.1 the leakage-resilient storage scheme randomly chooses s_L and then computes s_R such that $s_L \cdot s_R = s$. We define the tuple leakage parameter $\boldsymbol{\lambda} = (\lambda_1, \lambda_2)$ as follows: $\boldsymbol{\lambda}$-limited adversary $\mathcal{A}$ sends a leakage function $\mathbf{f} = (f_{1j}, f_{2j})$ and obtains at most λ_1, λ_2 amount of

leakage from each of the two encodings of the secret s respectively: $f_{1j}(s_L)$ and $f_{2j}(s_R)$.

As mentioned in Definition 2.4.4, the leakage-resilient storage scheme can continuously refresh the encodings of the secret. Therefore, after executing the refreshing protocol it outputs new random-looking encodings of the same secret. So for the $\boldsymbol{\lambda}$-limited adversary again the situation is as before. Thus, refreshing the encodings will help to obtain leakage resilience over a number of protocol executions.

The computation of exponentiations is also split into two parts. Let $\mathbb{G}$ be a group of prime order q with generator g. Let $s \xleftarrow{\$} \mathbb{Z}_q^*$ be a long-term secret key and $E = g^e$ be a received ephemeral value. Then, the value Z needs to be computed as $Z \leftarrow E^s$. In the leakage-resilient setting, in the initial setup the secret key is encoded as $s_L, s_R \leftarrow \text{Encode}_{\mathbb{Z}_q^*}^{n,1}(s)$. So the vector $s_L = (s_{L1}, \cdots, s_{Ln})$ and the vector $s_R = (s_{R1}, \cdots, s_{Rn})$ are such that $s = s_{L1}s_{R1} + \cdots + s_{Ln}s_{Rn}$. Then the computation of E^s can be performed as two component-wise computations as follows: compute the intermediate vector $T \leftarrow E^{s_L} = (E^{s_{L1}}, \cdots, E^{s_{Ln}})$ and then compute the element $Z \leftarrow T^{s_R} = E^{s_{L1}s_{R1}} E^{s_{L2}s_{R2}} \cdots E^{s_{L1}s_{R1}} = E^{s_{L1}s_{R1} + \cdots + s_{Ln}s_{Rn}} = E^s$.

Protocol Construction

Using the above ideas, by encoding the secret using a leakage-resilient storage scheme, and refreshing the encoded secret using a refreshing protocol, it is possible to hide the secret from a $\boldsymbol{\lambda}$-limited adversary. Further, it is possible to successfully compute the exponentiation using the encoded secrets. We now use these primitives to construct a CAFL-eCK-secure key exchange protocol, using an eCK-secure key exchange protocol as an underlying primitive.

Let $\Lambda_{\mathbb{Z}_q^*}^{n,1} = (\text{Encode}_{\mathbb{Z}_q^*}^{n,1}, \text{Decode}_{\mathbb{Z}_q^*}^{n,1})$ be the leakage-resilient storage scheme which is used to encode secret keys and $\text{Refresh}_{\mathbb{Z}_q^*}^{n,1}$ be the $(\ell, \boldsymbol{\lambda}, \epsilon)$-secure leakage-resilient refreshing protocol of $\Lambda_{\mathbb{Z}_q^*}^{n,1}$.

As we can see, the obvious way of key generation (initial setup) in a protocol principal of this protocol is as follows: first pick $a \xleftarrow{\$} \mathbb{Z}_q^*$ as the long-term secret key, then encode the secret key as $(a_L^0, a_R^0) \leftarrow \text{Encode}_{\mathbb{Z}_q^*}^{n,1}(a)$, then compute the long-term public key $A = g^a$ using the two encodings (a_L^0, a_R^0), and finally erase a from the memory. The potential threat to that key generation mechanism is that even though the long-term secret key a is erased from the memory, it might not be properly erased and can be leaked to the adversary during the key

generation. In order to avoid such a vulnerability, we randomly picks two values $a_L^0 \xleftarrow{\$} (\mathbb{Z}_q^*)^n \backslash \{(0^n)\}$, $a_R^0 \xleftarrow{\$} (\mathbb{Z}_q^*)^{n\times 1} \backslash \{(0^{n\times 1})\}$ and use them as the encodings of the long-term secret key a of a protocol principal. As explained earlier, we use a_L^0, a_R^0 to compute the corresponding long-term public key A in two steps as $a' \leftarrow g^{a_L^0}$ and $A \leftarrow a'^{a_R^0}$. Thus, it is possible to avoid exposing the un-encoded secret key a at any point of time in the key generation and hence avoid leaking directly from a at the key generation step. Further, the random vector a_L^0 is multiplied with the random vector a_R^0, such that $a = a_L^0 \cdot a_R^0$, which will give a random integer a in the group $\mathbb{Z}_q^*$. Therefore, this approach is same as picking $a \xleftarrow{\$} \mathbb{Z}_q^*$ at first and then encode, but in the reverse order. During the protocol execution both a_L^0, a_R^0 are continuously refreshed and refreshed encodings a_L^j, a_R^j are used to exponentiation computations.

Table 5.2 shows the protocol P2. In this setting leakage of a long-term secret key does not happen directly from the long-term secret key itself, but from the two encodings of the long-term secret key (the leakage function $\mathbf{f} = (f_{1j}, f_{2j})$ directs to the each individual encoding). During the exponentiation computations and the refreshing operation collectively at most $\boldsymbol{\lambda} = (\lambda_1, \lambda_2)$ leakage is allowed to the adversary from each of the two portions independently. Then, the two portions of the encoded long-term secret key are refreshed and in the next protocol session another $\boldsymbol{\lambda}$-bounded leakage is allowed. Thus, continuous leakage is allowed.

Alice (Initiator)		Bob (Responder)
	Initial Setup	
$a_L^0 \xleftarrow{\$} (\mathbb{Z}_q^*)^n \backslash \{(0^n)\}, a_R^0 \xleftarrow{\$} (\mathbb{Z}_q^*)^{n\times 1} \backslash \{(0^{n\times 1})\}$ $a' \leftarrow g^{a_L^0}, A \leftarrow (a')^{a_R^0}$		$b_L^0 \xleftarrow{\$} (\mathbb{Z}_q^*)^n \backslash \{(0^n)\}, b_R^0 \xleftarrow{\$} (\mathbb{Z}_q^*)^{n\times 1} \backslash \{(0^{n\times 1})\}$ $b' \leftarrow g^{b_L^j}, B \leftarrow (b')^{b_R^0}$
	Protocol Execution	
$x \xleftarrow{\$} \mathbb{Z}_q^*, X \leftarrow g^x$	$\xrightarrow{Alice, X}$ $\xleftarrow{Bob, Y}$	$y \xleftarrow{\$} \mathbb{Z}_q^*, Y \leftarrow g^y$
$T_1 \leftarrow B^{a_L^j}, Z_1 \leftarrow T_1^{a_R^j}$		$T_3 \leftarrow A^{b_L^j}, Z_1' \leftarrow T_3^{b_R^j}$
$Z_2 \leftarrow B^x$		$T_4 \leftarrow X^{b_L^j}, Z_2' \leftarrow T_4^{b_R^j}$
$T_2 \leftarrow Y^{a_L^j}, Z_3 \leftarrow T_2^{a_R^j}$		$Z_3' \leftarrow A^y$
$Z_4 \leftarrow Y^x$		$Z_4' \leftarrow X^y$
$(a_L^{j+1}, a_R^{j+1}) \leftarrow \text{Refresh}_{\mathbb{Z}_q^*}^{n,1}(a_L^j, a_R^j)$		$(b_L^{j+1}, b_R^{j+1}) \leftarrow \text{Refresh}_{\mathbb{Z}_q^*}^{n,1}(b_L^j, b_R^j)$
$K \leftarrow \text{H}(Z_1, Z_2, Z_3, Z_4, Alice, X, Bob, Y)$		$K \leftarrow \text{H}(Z_1', Z_2', Z_3', Z_4', Alice, X, Bob, Y)$
	K is the session key	

Table 5.2: Concrete construction of Protocol P2

5.4.2 Security Proof of the Protocol P2 in the CAFL-eCK Model

Theorem 5.4.1. If the underlying refreshing protocol $\mathrm{Refresh}^{n,1}_{\mathbb{Z}_q^*}$ is $(\ell, \boldsymbol{\lambda}, \epsilon)$-secure leakage-resilient refreshing protocol of the leakage-resilient storage scheme $\Lambda^{n,1}_{\mathbb{Z}_q^*}$ and the underlying key exchange protocol P1 is eCK-secure key exchange protocol, then the protocol P2 is $\boldsymbol{\lambda} -$ CAFL-eCK-secure.

Let $\mathcal{A}$ be any PPT adversary against the key exchange protocol P2. Then the advantage of $\mathcal{A}$ against the CAFL-eCK-security of the protocol P2, $\mathrm{Adv}^{\boldsymbol{\lambda}-\text{CAFL-eCK}}_{\text{P2}}$ is:

$$\mathrm{Adv}^{\boldsymbol{\lambda}-\text{CAFL-eCK}}_{\text{P2}}(\mathcal{A}) \leq N_P\Big(\mathrm{Adv}^{\text{eCK}}_{\text{P1}}(\mathcal{A}) + \epsilon\Big) \ . \tag{5.61}$$

Proof. **Sequence of Games.**

Game 1. This is the original game.

Game 2. Same as the Game 1 with the following exception: before $\mathcal{A}$ begins, an identity of a random principal $U^* \xleftarrow{\$} \{U_1, \ldots, U_{N_P}\}$ is chosen. Challenger expects that the adversary will issue the `Test` for a session which involves the principal U^* ($\Pi^{\cdot}_{U^*,\cdot}$ or $\Pi^{\cdot}_{\cdot,U^*}$). If not the challenger aborts the game.

Game 3. Same as the Game 2 with the following exception: challenger picks a random $s \xleftarrow{\$} \mathbb{Z}_q^*$ and uses encodings of s to simulate the adversarial leakage queries $\mathbf{f} = (f_{1j}, f_{2j})$.

Differences between games.

In this section the adversary's advantage of distinguishing each game from the previous game is investigated. Let $\mathrm{Adv}_{\text{Game } x}(\mathcal{A})$ denotes the advantage of the adversary $\mathcal{A}$ winning the Game x.

Game 1 is the original game. Hence,

$$\mathrm{Adv}_{\text{Game 1}}(\mathcal{A}) = \mathrm{Adv}^{\boldsymbol{\lambda}-\text{CAFL-eCK}}_{\text{P2}}(\mathcal{A}) \ . \tag{5.62}$$

Game 1 and Game 2. The probability of Game 2 to be halted due to incorrect choice of the test session is $1 - \frac{1}{N_P}$. Unless the incorrect choice happens, Game 2 is identical to Game 1. Hence,

$$\mathrm{Adv}_{\mathrm{Game\ 2}}(\mathcal{A}) = \frac{1}{N_P}\mathrm{Adv}_{\mathrm{Game\ 1}}(\mathcal{A}) \ . \tag{5.63}$$

Game 2 and Game 3. We construct an algorithm $\mathcal{B}$ against a leakage-resilient refreshing protocol challenger of $\mathrm{Refresh}^{n,1}_{\mathbb{Z}^*_q}$, using the adversary $\mathcal{A}$ as a subroutine.

The $(\ell, \boldsymbol{\lambda}, \epsilon) - \mathrm{Refresh}^{n,1}_{\mathbb{Z}^*_q}$ refreshing protocol challenger chooses $s_0, s_1 \xleftarrow{\$} \mathbb{Z}^*_q$ and sends them to the algorithm $\mathcal{B}$. Further, the refreshing protocol challenger randomly chooses $s \xleftarrow{\$} \{s_0, s_1\}$ and uses s as the secret to compute the leakage from encodings of s. Let $\boldsymbol{\lambda} = (\lambda_1, \lambda_2)$ be the leakage bound and assume the refreshing protocol challenger continuously refreshes the two encodings of the secret s.

When the algorithm $\mathcal{B}$ gets the challenge of s_0, s_1 from the refreshing protocol challenger, $\mathcal{B}$ uses s_0 as the secret key of the protocol principal U^* and computes the corresponding public key. For all other protocol principals $\mathcal{B}$ sets secret/public key pairs by itself. Using the setup keys, $\mathcal{B}$ computes answers to all the queries from $\mathcal{A}$ and simulates the view of CAFL-eCK challenger of the protocol P2. $\mathcal{B}$ computes the leakage of secret keys by computing the adversarial leakage function $\mathbf{f}$ on the corresponding secret key (encodings of secret key), except the secret key of the protocol principal U^*. In order to obtain the leakage of the secret key of U^*, algorithm $\mathcal{B}$ queries the the refreshing protocol challenger with the adversarial leakage function $\mathbf{f}$, and passes that leakage to $\mathcal{A}$.

If the secret s chosen by the refreshing protocol challenger is s_0, the leakage of the secret key of U^* simulated by $\mathcal{B}$ (with the aid of the refreshing protocol challenger) is the real leakage. Then the simulation is identical to Game 2. Otherwise, the leakage of the secret key of U^* simulated by $\mathcal{B}$ is a leakage of a random value. Then the simulation is identical to Game 3. Hence,

$$|\mathrm{Adv}_{\mathrm{Game\ 2}}(\mathcal{A}) - \mathrm{Adv}_{\mathrm{Game\ 3}}(\mathcal{A})| \leq \epsilon \ . \tag{5.64}$$

Game 3. Since the leakage is computed using a random s value, the adversary $\mathcal{A}$ will not get any advantage due to the leakage. Therefore, the advantage $\mathcal{A}$ will get is same as the advantage that $\mathcal{A}$ has against an eCK challenger of the protocol P1, because both P1 and P2 are effectively doing the same computation, regardless of the leakage-resilient exponentiation computation in the protocol P2, and with no useful leakage the CAFL-eCK model is same as the eCK model.

Hence,

$$\mathrm{Adv}_{\text{Game 3}}(\mathcal{A}) = \mathrm{Adv}^{\mathrm{eCK}}_{\mathrm{P1}}(\mathcal{A}) \ . \tag{5.65}$$

We find,

$$\mathrm{Adv}^{\boldsymbol{\lambda}-\text{CAFL-eCK}}_{\mathrm{P2}}(\mathcal{A}) \leq N_P\Big(\mathrm{Adv}^{\mathrm{eCK}}_{\mathrm{P1}}(\mathcal{A}) + \epsilon\Big) \ . \tag{5.66}$$

□

5.4.3 Leakage Tolerance of the Protocol P2

The order of the group $\mathbb{G}$ is q. Let $m = 1$ in the leakage-resilient storage scheme $\Lambda^{n,1}_{\mathbb{Z}^*_q}$. According to the Lemma 2.4.1, if $m < n/20$, then the leakage parameter for the leakage-resilient storage scheme is $\boldsymbol{\lambda}_\Lambda = (0.3n \log q, 0.3n \log q)$. Let $n = 21$, then $\boldsymbol{\lambda}_\Lambda = (6.3 \log q, 6.3 \log q)$ bits. According to the Theorem 2.4.1, if $m/3 \leq n$ and $n \geq 16$, the refreshing protocol $\mathrm{Refresh}^{n,1}_{\mathbb{Z}^*_q}$ of the leakage-resilient storage scheme $\Lambda^{n,1}_{\mathbb{Z}^*_q}$ is tolerant to (continuous) leakage up to $\boldsymbol{\lambda}_{\mathrm{Refresh}} = \boldsymbol{\lambda}_\Lambda/2 = (3.15 \log q, 3.15 \log q)$ bits, per occurrence.

When a secret key s (of size $\log q$ bits) of the protocol P2 is encoded into two parts, the left part s_L will be $n \cdot \log q = 21 \log q$ bits and the right part s_R will be $n \cdot 1 \cdot \log q = 21 \log q$ bits. For a tuple leakage function $\mathbf{f} = (f_{1j}, f_{2j})$ (each leakage function $f_{(\cdot)}$ for each of the two parts s_L and s_R), there exists a tuple leakage bound $\boldsymbol{\lambda} = (\lambda, \lambda)$ for each leakage function $f_{(\cdot)}$, such that $\lambda = 3.15 \log q$ bits, per occurrence, which is $\frac{3.15 \log q}{21 \log q} \times 100\% = 15\%$ of the size of a part. The overall leakage amount is unbounded since continuous leakage is allowed.

5.5 Summary

We have presented a *generic security model*, namely the (·)AFL-eCK model, improving the amount and type of secret leakage that can be tolerated in authenticated key exchange protocols. Our generic model allows the adversary to fully compromise a variety of long-term and short-term ephemeral values, as well as obtain partial, adaptive, *either bounded or continuous*, after-the-fact leakage of long-term secret keys. Previous key exchange security models either do not consider partial leakage at all (BR, CK, eCK) or allow only leakage before the test session is queried and none after (MO). Even the CAFL model we presented

in Chapter 4 enforces more restrictions to the freshness definition than the generic (·)AFL-eCK model in this chapter. The generic (·)AFL-eCK model captures a wide variety of practical attack scenarios, including side channel attacks. Further, the model gives flexibility to choose either bounded or continuous leakage setting and available underlying primitives, and it provides a security framework to analyze after-the-fact leakage-resilient key exchange protocols. We have given a generic protocol, secure in a weaker variant of our generic model, that relies on a CPLA2-secure $\epsilon -$ PG-IND-public-key cryptosystem and an UFCMLA-secure signature scheme.

Further, we presented a concrete construction of a CAFL-eCK-secure key exchange protocol by using a leakage-resilient storage scheme and its refreshing protocol. The main technique used to achieve after-the-fact leakage resilience is encoding the secret key into two parts and only allowing the independent leakage from each part.

5.6 Comparison of Key Exchange Security Models and Protocols

In this thesis, we have presented two security models for key exchange protocols, addressing more granular partial leakage of long-term secret keys, namely continuous after-the-fact leakage model (CAFL) and the generic after-the-fact leakage eCK model ((·)AFL-eCK) (and w(·)AFL-eCK) model). Further, we presented generic protocol constructions for each of CAFL and w(·)AFL-eCK) models and a concrete protocol construction for the continuous leakage variant of the (·)AFL-eCK model, the CAFL-eCK model.

5.6.1 Comparison of Security Models

Table 5.3 summarizes the adversarial powers of the two instantiations of the generic (·)AFL-eCK model and the CAFL model, in comparison with the adversarial powers of the eCK model [LLM07] and the Moriyama–Okamoto (MO) model [MO11]. There are four `Corrupt`–`EphemeralKeyReveal` query combinations which do not trivially expose the session key. In the column "Combinations" of Table 5.3, we mention how many of them are allowed in the corresponding security model. We discussed query combinations in detail in Section 4.2.3. The $*$ indicates that

the w(·)AFL-eCK model does not allow the `EphemeralKeyReveal` query to reveal the randomness used in the underlying signature scheme.

Security model	Combinations	Leakage model	After-the-fact
eCK [LLM07]	4/4	No	No
MO [MO11]	4/4	Bounded	No
CAFL (Chapter 4)	2/4	Continuous	Yes
(·)AFL-eCK (Chapter 5)	4/4	Bounded/Continuous	Yes
w(·)AFL-eCK (Chapter 5)	4*/4	Bounded/Continuous	Yes

Table 5.3: Key exchange security models with reveal queries and leakage allowed

The eCK model is a non-leakage security model and Moriyama and Okamoto have constructed a leakage security model based on eCK model. The Moriyama-Okamoto model allows bounded leakage, only before the target session is established. We constructed the CAFL model, allowing continuous leakage even after the target session is established, while enforcing additional restrictions to the eCK-style freshness condition. Therefore, the strength of the CAFL model is not directly comparable with eCK or Moriyama-Okamoto models, but the CAFL model clearly allows more granular partial leakage. Then we constructed the generic (·)AFL-eCK, releasing the additional restrictions to the freshness condition which had been introduced in the CAFL model. Thus, the two instantiations of the generic (·)AFL-eCK model, namely the BAFL-eCK and CAFL-eCK models are stronger than the eCK, the Moriyama-Okamoto and the CAFL models.

5.6.2 Comparison of Key Exchange Protocols

The Table 5.4 compares the protocol $\pi 1$ of Chapter 4, the protocol π of Chapter 5 and the protocol P2 of Chapter 5, with the NAXOS protocol [LLM07] and the Moriyama–Okamoto protocol [MO11].

Protocol	Security Model	Assumptions	Construction
NAXOS [LLM07]	eCK	GDH, RO	Concrete
MO [MO11]	MO	DDH, HPS, PRF, (λ, ϵ)-Ext	Concrete
$\pi 1$ of Chapter 4	CAFL	CCLA2-secure PKE, PRF,	Generic
π of Chapter 5	w(·)AFL-eCK	DDH, ODH, ϵ – PG-IND and CPLA2-secure PKE, secure KDF, PRF	Generic
P2 of Chapter 5	CAFL-eCK	GDH, RO	Concrete

Table 5.4: Comparison of key exchange protocols

The NAXOS protocol is the first concrete protocol which is proven secure in the eCK model. Being an eCK-secure protocol, NAXOS does not provide any security

guarantee for side-channel attacks. The Moriyama-Okamoto protocol is the first concrete protocol which is proven secure in a leakage security model, namely the Moriyama-Okamoto model. The Moriyama-Okamoto protocol is resistant to a bounded amount of leakage of long-term secret keys only before the target session is activated. We constructed a generic CAFL-secure protocol, which can be instantiated using any suitable leakage-resilient public-key encryption scheme, in a way that it is secure against continuous leakage of long-term secret keys even after the target session is activated. Then we constructed a generic w(·)AFL-eCK-secure protocol in a way that it is possible to instantiate a wBAFL-eCK or wCAFL-eCK-secure key exchange protocol using any suitable leakage-resilient public-key encryption scheme and a leakage-resilient signature scheme. Since there are currently no suitable leakage-resilient public-key encryption schemes to instantiate the continuous leakage-resilient variant of the generic protocol, we constructed a concrete CAFL-eCK-secure protocol, namely protocol P2, using leakage-resilient storage schemes. Protocol P2 is proven secure in the strongest leakage-security model for key exchange, guaranteeing the eCK-style security as well as tolerance against continuous leakage of long-term secret keys even after the target session is activated.

The generic CAFL-secure protocol of Chapter 4 can be instantiated with the CCLA2-secure public-key encryption scheme of Dziemboski and Faust [DF11], whereas the generic w(·)AFL-eCK-secure protocol of Chapter 5 can be instantiated as a wBAFL-eCK-secure protocol using the CPLA2-secure pair generation indistinguishable public-key encryption scheme of Halevi and Lin [HL11] and the UFCMLA-secure signature scheme of Katz and Vaikuntanathan [KV09]. Table 5.5 compares these protocol instantiations and the protocol P2 with the NAXOS protocol [LLM07] and the Moriyama-Okamoto (MO) protocol [MO11], in terms of computation cost and the security model.

Protocol	**Initiator Cost**	**Responder Cost**	**Security Model**
NAXOS [LLM07]	4 **Exp**	4 **Exp**	eCK
MO [MO11]	8 **Exp**	8 **Exp**	MO
π1 of Chapter 4 instantiation	10 **Exp**	10 **Exp**	CAFL
π of Chapter 5 instantiation	12 **Exp**	12 **Exp**	wBAFL-eCK
P2 protocol of Chapter 5	6 **Exp**	6 **Exp**	CAFL-eCK

Table 5.5: Security and efficiency comparison of key exchange protocols

Generally, while adding security features we have to sacrifice the computation cost. The same thing happens when coming from the NAXOS protocol to the

protocol π of Chapter 5. But we can see an exception when coming to the protocol P2 from the protocol π of Chapter 5. The reason for this is that we use the leakage-resilient storage scheme to store the secret values and only rearrange the way we do the necessary exponentiation computation. By contrast in the $\pi 1$ and π constructions we use the whole Enc/Dec operation of the leakage-resilient public-key encryption scheme, which is a combination of several exponentiation operations and hence more expensive.

Chapter 6

Compression-based Leakage

Contents

Compression is desirable for network applications as it saves bandwidth; however, when data is compressed before being encrypted, the amount of compression leaks information about the amount of redundancy in the plaintext. This side channel has led to the successful CRIME and BREACH attacks on web traffic protected by the Transport Layer Security (TLS) protocol. The general guidance in light of these attacks has been to disable compression, preserving confidentiality but sacrificing bandwidth. We examine two techniques—heuristic separation of secrets and fixed-dictionary compression—for enabling compression while protecting high-value secrets, such as cookies, from attack. We model the security offered by these techniques and report on the amount of compressibility that they can achieve.

6.1 Introduction

To save communication costs, network applications often compress data before transmitting it; for example, the Hypertext Transport Protocol (HTTP) [FR14, §4.2] has an optional mechanism in which a server compresses the body of an HTTP response, most commonly using the gzip algorithm. When encryption is used to protect communication, compression must be applied before encryption (since ciphertexts should look random, they should have little apparent redundancy that can be compressed). In fact, to facilitate this, the Transport Layer Security (TLS) protocol [DR08, §6.2.2] has an optional compression mode that will compress all application data before encrypting it. While compression is useful for reducing the size of transmitted data, it has had a negative impact when combined with encryption, because the amount of compression acts as a *side channel*.

Compression-based Leakage. In 2002, Kelsey [Kel02] showed how compression can act as a form of side-channel leakage. If plaintext data is compressed

before being encrypted, the length of the ciphertext reveals information about the amount of compression, which in turn can reveal information about the plaintext. Kelsey notes that this side channel differs from other types of side channels in two key ways: "it reveals information about the plaintext, rather than key material", and "it is a property of the algorithm, not the implementation".

Kelsey's most powerful attack is an *adaptive chosen input attack*: if an attacker is allowed to choose inputs x that are combined with a target secret s and the concatenation $x\|s$ is compressed and encrypted, observing the length of the outputs can eventually allow the attacker to extract the secret s. For example, to determine the first character of s, the attacker could ask to have the string $x =$ `prefix*prefix` combined with s, then compressed and encrypted, for every possible character `*`; in one case, when `*` $= s_1$, the amount of redundancy is higher and the ciphertext should be shorter. Once each character of s is found, the attack can be carried out on the next character.

Key to this attack is the fact that most compression algorithms (such as the DEFLATE algorithm underlying gzip) are *adaptive*: they adaptively build and maintain a *dictionary* of recently observed strings, and replace subsequent occurrences of that string with a code.

The CRIME and BREACH Attacks. In 2012, Rizzo and Duong [RD12] showed how to apply Kelsey's adaptive chosen input attack against gzip compression as used in TLS, in what they called the *Compression Ratio Info-leak Mass Exploitation (CRIME)* attack. The primary target of the CRIME attack was the user's cookie in the HTTP header. If the victim visited an attacker-controlled web page, the attacker could use Javascript to cause the victim to send HTTP requests to URLs of the attacker's choice on a specified server. The attacker could adaptively choose those URLs to include a prefix to carry out Kelsey's adaptive chosen input attack. Some care is required to ensure the padding does not hide the length with block ciphers, but this can be dealt with. The CRIME attack also applies to compression as used in the SPDY protocol [The09].

Trustworthy Internet Movement's SSL Pulse report for June 2015 finds that just 4.8% of websites have TLS compression enabled and vulnerable to the CRIME attack [Tru15]; moreover, all major browsers have disabled it.

However, compression is also built into the HTTP protocol: servers can optionally compress the body of HTTP responses. While this excludes the cookie in the header, this attack can still succeed against secret values in the

HTTP body, such as anti-cross-site request forgery (CSRF) tokens. Suggested by Rizzo and Duong, this was demonstrated by Gluck et al. [GHP13] in the *Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext (BREACH)* attack.

Mitigation Techniques. Gluck et al. [GHP13] discussed several possible mitigation techniques against the BREACH attack, listed in decreasing order of effectiveness:

1. Disabling HTTP compression
2. Separating secrets from user input
3. Randomizing secrets per request
4. Masking secrets (effectively randomizing by XORing with a random nonce)
5. Length hiding (by adding a random number of bytes to the responses)
6. Rate-limiting the requests

Despite the demonstrated practicality of the BREACH attack, support for and use of HTTP compression remains widespread, due in large part to the value of decreasing communication costs and time. In fact, compression is even more tightly integrated into the proposed HTTP version 2 [BPT14] than previous versions. Techniques 2–4 generally require changes to both browsers and web servers. For example, masking secrets such as anti-CSRF tokens requires new mark-up for secrets, which browsers and servers can interpret to apply the randomized masking technique. Techniques 5–6 can be unilaterally applied by web servers, though length hiding can be defeated with statistical averaging, and rate-limiting must find a balance between legitimate requests and information leakage.

Despite these attacks, Klinc et al. [KHJ+10] demonstrated that block ciphers operating in various chaining modes can be compressed without compromising the security of the encryption scheme. Differently, Peon and Ruellan [PR15] introduced a technique of header compression for HTTP/2, namely HPACK. It compresses header fields independently from other inputs, and after the compression resets the dictionary. So each header is compressed in its own context.

There has been little academic study of compression and encryption. Besides Kelsey's adaptive chosen input attack and the related CRIME and BREACH attacks, the only relevant work we are aware of is that of Kelley and Tamassia [KT14]. They give a new security notion called *entropy-restricted semantic security* (ER-IND-CPA) for *keyed compression functions* which combine both encryption and compression: compared with the normal indistinguishability under chosen plaintext attack (IND-CPA) security notion, in ER-IND-CPA the adversary should not be able to distinguish between the encryption of two messages that *compress* to the same length. Kelley and Tamassia then show how to construct a cipher based on the LZW compression algorithm by rerandomizing the compression dictionary. Unfortunately, the ER-IND-CPA notion does not capture the CRIME and BREACH attacks, which depend on observing messages that compress to different lengths.

In leakage-resilient security definitions [AGV09, NS09], leakage of the secret key is addressed. This differs from the setting in compression-based side-channel attacks, which addresses leakage of the plaintext. Thus, previous leakage-resilient approaches are not suitable to model compression-based side-channel attacks.

6.2 Preliminaries

Notation. If x is a string, then x_i denotes the ith character of x; $x_{i:\ell}$ denotes the length-ℓ substring of x starting at position i: $x_{i:\ell} = x_i \| \ldots \| x_{i+\ell-1}$. If x and y are strings, then $x \preceq y$ denotes that x is a substring of y. The *index* of x in y is the smallest i such that $y_{i:|x|} = x$ and is denoted by $\mathrm{ind}_y(x)$; if $x \not\preceq y$, we denote $\mathrm{ind}_y(x) = \perp$. The empty string is denoted by ϵ.

6.2.1 Encryption and Compression Schemes

Recall the standard definition of an encryption scheme:

Definition 6.2.1 (Symmetric-key encryption)**.** A *symmetric-key encryption scheme* Π for message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$ is a tuple of algorithms:

- $\mathrm{KeyGen}() \xrightarrow{\$} k$: A probabilistic *key generation algorithm* that generates a random key k in the keyspace $\mathcal{K}$.
- $\mathrm{Enc}_k(m) \xrightarrow{\$} c$: A possibly probabilistic *encryption algorithm* that takes as input a key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$ and outputs a ciphertext $c \in \mathcal{C}$.

- $\mathrm{Dec}_k(c) \to m'$ or $\perp$: A deterministic *decryption algorithm* that takes as input a key $k \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$, and outputs either a message $m' \in \mathcal{M}$ or an error symbol $\perp$.

Correctness of symmetric-key encryption is defined in the obvious way: for all $k \xleftarrow{\$} \mathrm{KeyGen}()$ and all $m \in \mathcal{M}$, we require that $\mathrm{Dec}_k(\mathrm{Enc}_k(m)) = m$.

Definition 6.2.2 (Compression scheme)**.** A *compression scheme* Γ for message space $\mathcal{M}$ with output space $\mathcal{O}$ is a pair of algorithms:

- $\mathrm{Comp}(m) \xrightarrow{\$} o$: A possibly probabilistic *compression* algorithm that takes as input a message $m \in \mathcal{M}$ and outputs an encoded value $o \in \mathcal{O}$.
- $\mathrm{Decomp}(o) \to m'$ or $\perp$: A *decompression* algorithm that takes as input an encoded value $o \in \mathcal{O}$ and outputs a message $m' \in \mathcal{M}$ or an error symbol $\perp$.

Note that $|\mathrm{Comp}(m)|$ may not necessarily be less than $|m|$; Shannon's coding theorem [Cas00] implies that no algorithm can encode every message with shorter length, so not all messages may actually be "compressed": some may increase in length.

Correctness of a compression scheme is again defined in the obvious way: for all $m \in \mathcal{M}$, we require that $\mathrm{Decomp}(\mathrm{Comp}(m)) = m$.

We are interested in *symmetric-key compression-encryption schemes*, which formally are just symmetric-key encryption schemes as in Definition 6.2.1, but usually have the goal of outputting shorter ciphertexts via some form of compression. Of course, every symmetric-key encryption scheme is also a symmetric-key compression-encryption scheme, with "compression" being the identity function. We will often deal with the following specific, natural composition of compression and symmetric-key encryption:

Definition 6.2.3 (Composition of compression and encryption)**.** Let $\Gamma = (\mathrm{Comp}, \mathrm{Decomp})$ be a compression scheme with message space $\mathcal{M}$ and output space $\mathcal{O}$. Let $\Pi = (\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec})$ be a symmetric-key encryption scheme with message space $\mathcal{O}$ and ciphertext space $\mathcal{C}$. The symmetric-key compression-encryption scheme $\Pi \circ \Gamma$ constructed from Γ and Π is the following tuple:

$$
\begin{aligned}
(\Pi \circ \Gamma).\mathrm{KeyGen}() &= \Pi.\mathrm{KeyGen}() \\
(\Pi \circ \Gamma).\mathrm{Enc}_k(m) &= \Pi.\mathrm{Enc}_k(\Gamma.\mathrm{Comp}(m)) \\
(\Pi \circ \Gamma).\mathrm{Dec}_k(c) &= \Gamma.\mathrm{Decomp}(\Pi.\mathrm{Dec}_k(c))
\end{aligned}
$$

$\underline{\mathrm{Exp}^{\text{IND-CPA}}_{\Pi}(\mathcal{A})}$

$k \xleftarrow{\$} \Pi.\mathrm{KeyGen}()$
$b \xleftarrow{\$} \{0,1\}$
$(m_0, m_1, st) \xleftarrow{\$} \mathcal{A}^{E}()$
if $|m_0| \neq |m_1|$, **then return** $\perp$
$c \leftarrow \Pi.\mathrm{Enc}_k(m_b)$
$b' \xleftarrow{\$} \mathcal{A}^{E}(c, st)$
return $(b' = b)$

$\underline{E(m)}$

return $\Pi.\mathrm{Enc}_k(m)$

$\underline{\mathrm{Exp}^{\text{ER-IND-CPA}}_{\Pi,\mathcal{L}}(\mathcal{A})}$

$k \xleftarrow{\$} \Pi.\mathrm{KeyGen}()$
$b \xleftarrow{\$} \{0,1\}$
$(m_0, m_1, st) \xleftarrow{\$} \mathcal{A}^{E}()$
if $m_0 \notin \mathcal{L}$ or $m_1 \notin \mathcal{L}$, **then return** $\perp$
$c \leftarrow \Pi.\mathrm{Enc}_k(m_b)$
$b' \xleftarrow{\$} \mathcal{A}^{E}(c, st)$
return $(b' = b)$

$\underline{E(m)}$

return $\Pi.\mathrm{Enc}_k(m)$

Figure 6.1: Security experiments for indistinguishability under chosen plaintext attack (IND-CPA, left) and entropy-restricted IND-CPA (ER-IND-CPA, right)

Note that $\Pi \circ \Gamma$ is itself a symmetric-key encryption scheme with message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$. If Γ and Π are both correct, then so is $\Pi \circ \Gamma$.

6.2.2 Existing Security Notions

The standard security notion for symmetric-key encryption is indistinguishability of encrypted messages. Here we focus on chosen plaintext attack. The security experiment $\mathrm{Exp}^{\text{IND-CPA}}_{\Pi}(\mathcal{A})$ for indistinguishability under chosen plaintext attack (IND-CPA) of a symmetric-key encryption scheme Π against a stateful adversary $\mathcal{A}$ is given in Figure 6.1. The *advantage* of $\mathcal{A}$ in breaking the IND-CPA experiment for Π is $\mathrm{Adv}^{\text{IND-CPA}}_{\Pi}(\mathcal{A}) = \left|2 \Pr\left(\mathrm{Exp}^{\text{IND-CPA}}_{\Pi}(\mathcal{A}) = 1\right) - 1\right|$.

Kelley and Tamassia [KT14] give a definition of *entropy-restricted* IND-CPA *security* which applies to keyed compression schemes Π, and demands indistinguishability of encryptions of messages from the same class $\mathcal{L} \subseteq \mathcal{M}$; typically, $\mathcal{L}$ is the class of messages that encrypt (compress) to the same length under $\Pi.\mathrm{Enc}$, such as:

$$\mathcal{L}_\ell = \{m \in \mathcal{M} : |\Pi.\mathrm{Enc}(m)| = \ell\} \quad .$$

The ER-IND-CPA security experiment is given in Figure 6.1; the corresponding advantage is defined similarly. Kelley and Tamassia note that any IND-CPA-secure symmetric-key encryption scheme Π, combined with any compression scheme Γ, is immediately ER-IND-CPA-secure. As well, it is easily seen that if a symmetric-key encryption scheme is ER-IND-CPA-secure for the class $\mathcal{L}_\ell = \{m \in \mathcal{M} : |m| = \ell\}$, then that scheme is also an IND-CPA-secure symmetric-key

encryption.

6.2.3 New Security Notions

We focus on the ability of an attacker to learn about a secret piece of data inside a larger piece of data, where the attacker controls everything except the secret data. We use the term *cookie* to refer to the secret data; in practice, this could be an HTTP cookie in a header, an anti-CSRF token, or some piece of personal information. We will allow the attacker to adaptively obtain encryptions of compressions of data of the form $m' \| ck \| m''$ for a secret cookie ck and adversary-chosen message prefix m' and suffix m''.

We now present three notions for the security of cookies in the context of compression-encryption schemes:

- *Cookie recovery* (CR) *security*: A simple, but relatively weak, security notion for symmetric-key compression-encryption schemes: it should be hard for the attacker to *fully recover* a secret value, even given adaptive access to an oracle that encrypts plaintexts of its choosing with the target cookie embedded.

- *Random cookie indistinguishability* (RCI) *security*: The adversary has to decide which of two randomly chosen cookies was embedded in the encrypted plaintext, given adaptive access to an oracle that encrypts plaintexts of its choosing with the target cookie embedded.

- *Chosen cookie indistinguishability* (CCI) *security*: Here, the adversary has to decide which of two cookies of the adversary's choice was embedded in the encrypted plaintext, given adaptive access to an oracle that encrypts plaintexts of its choosing with the target cookie embedded.

These security notions are formalized in the following definition, which refers to the security experiments shown in Figure 6.2.

Definition 6.2.4 (CR, RCI, CCI security)**.** Let Ψ be a symmetric-key compression-encryption scheme. Let $\mathcal{A}$ denote an algorithm. Let $\mathcal{CK}$ denote the cookie space. Let $\mathsf{xxx} \in \{\mathrm{CR}, \mathrm{RCI}, \mathrm{CCI}\}$ be a security notion. Consider the security experiment $\mathrm{Exp}^{\mathsf{xxx}}_{\Psi,\mathcal{CK}}(\mathcal{A})$ in Figure 6.2. Define $\mathrm{Adv}^{\mathrm{CR}}_{\Psi,\mathcal{CK}}(\mathcal{A}) = \Pr\left(\mathrm{Exp}^{\mathrm{CR}}_{\Psi,\mathcal{CK}}(\mathcal{A}) = 1\right)$ as the probability that $\mathcal{A}$ wins the cookie recovery experiment for Ψ and $\mathcal{CK}$.

$\underline{\mathrm{Exp}^{\mathrm{CR}}_{\Psi,\mathcal{CK}}(\mathcal{A})}$

1: $k \xleftarrow{\$} \Psi.\mathrm{KeyGen}()$
2: $ck \xleftarrow{\$} \mathcal{CK}$
3: $ck' \xleftarrow{\$} \mathcal{A}^{E_1,E_2}()$
4: **return** $(ck' = ck)$

$\underline{E_1(m', m'')}$

1: **return** $\Psi.\mathrm{Enc}_k(m'\|ck\|m'')$

$\underline{E_2(m)}$

1: **return** $\Psi.\mathrm{Enc}_k(m)$

$\underline{\mathrm{Exp}^{\mathrm{RCI/CCI}}_{\Psi,\mathcal{CK}}(\mathcal{A})}$

1: $k \xleftarrow{\$} \Psi.\mathrm{KeyGen}()$
2: **if** RCI **then**
3: $\quad(ck_0, ck_1) \xleftarrow{\$} \mathcal{CK}$ s.t. $|ck_0| = |ck_1|$; $st \leftarrow \perp$
4: **else if** CCI **then**
5: $\quad(ck_0, ck_1, st) \xleftarrow{\$} \mathcal{A}^{E_2}()$ s.t. $|ck_0| = |ck_1|$
6: $b \xleftarrow{\$} \{0, 1\}$
7: $b' \xleftarrow{\$} \mathcal{A}^{E_1,E_2}(ck_0, ck_1, st)$
8: **return** $(b' = b)$

$\underline{E_1(m', m'')}$

1: **return** $\Psi.\mathrm{Enc}_k(m'\|ck_b\|m'')$

$\underline{E_2(m)}$

1: **return** $\Psi.\mathrm{Enc}_k(m)$

Figure 6.2: Security experiments for cookie recovery (left) and random cookie indistinguishability and chosen cookie indistinguishability (right) attacks

Similarly, define $\mathrm{Adv}^{\mathsf{xxx}}_{\Psi,\mathcal{CK}}(\mathcal{A}) = \left|2\Pr\left(\mathrm{Exp}^{\mathsf{xxx}}_{\Psi,\mathcal{CK}}(\mathcal{A}) = 1\right) - 1\right|$, $\mathsf{xxx} \in \{\mathrm{RCI}, \mathrm{CCI}\}$, as the advantage that $\mathcal{A}$ has in winning the random cookie and chosen cookie indistinguishability experiments.

Remark 6.2.1. The CR, RCI, and CCI security notions intentionally include only the confidentiality of the cookie as a security goal, and not the confidentiality of any non-cookie data in the rest of the message. In most applications it would be desirable to obtain confidentiality of non-cookie data as well, and in many real-world situations, the application layer's cookie and non-cookie data are jointly sent to the security layer (such as SSL/TLS) for encryption. Our notions do not preclude the scheme from encrypting the non-cookie data as well (and in fact our constructions in Sections 6.4 and 6.5 do so). However, it is not possible in general to require confidentiality of the non-cookie data while still allowing it to be compressed, as that brings us back around to the original problem that motivated the work—compression of adversary-provided data can lead to ciphertexts of different lengths that break indistinguishability. This cycle can be broken by demanding some length restriction on the separated non-cookie data, such as in the ER-IND-CPA notion described in Section 6.2.2, but we omit that complication to focus solely on the security of the high-value secret cookies.

6.3 Relations and Separations between Security Notions

Cookie recovery, being a computational problem rather than a decisional problem, is a weaker security notion. Keeping CR as an initial step, the RCI and CCI notions gradually increase the security afforded to the cookie.

The following relations exist between security notions for symmetric-key compression-encryption schemes:

$$\text{CCI} \implies \text{RCI} \implies \text{CR} \ .$$

In other words, every scheme that provides chosen cookie indistinguishability provides random cookie indistinguishability, and so on. Moreover, these notions are distinct, and we can show separations between them:

$$\text{CR} \not\Rightarrow \text{RCI} \not\Rightarrow \text{CCI} \ .$$

Additionally, we can connect our new notions with existing notions:

$$\text{ER-IND-CPA} \implies \text{IND-CPA} \implies \text{CCI} \qquad \text{and} \qquad \text{CCI} \not\Rightarrow \text{IND-CPA} \ .$$

These last relations should be interpreted as follows. A standard (non-compressing) IND-CPA-secure symmetric-key encryption scheme is also CCI-secure. This is not to say, however, that an IND-CPA-secure symmetric-key encryption scheme combined with a compression scheme, such as $\Pi \circ \Gamma$ in Definition 6.2.3, is CCI-secure.

Full proofs of these relations and separations are provided below, but first we provide a brief overview of each proof.

- IND-CPA $\implies$ CCI: A (non-compressing) IND-CPA-secure symmetric-key encryption scheme provides indistinguishability of any pair of equal-length chosen messages, including messages involving a cookie. The proof proceeds by a hybrid argument, making the cookie used in each query made by the adversary to its E_1 oracle independent of the secret bit b.

- CCI $\not\Rightarrow$ IND-CPA: A degenerate scheme that uses a separating-secrets filter (to be discussed in the Section 6.4) to extract secret cookies then

encrypt the cookies but not the non-cookie data is CCI-secure but not IND-CPA-secure for the whole message.

- CCI $\implies$ RCI: A straightforward simulation: an adversary who cannot distinguish between encryptions of equal-length cookies of its choosing can also not distinguish between encryptions of randomly chosen equal-length cookies.

- RCI $\not\implies$ CCI: A counterexample is constructed that uses a separating-secrets filter: an extra ciphertext component c_2 is added, consisting of a point function applied to the separated secrets, where the point function is 1 on a single, publicly known cookie value z. With high probability, two randomly chosen cookies will not match z, so c_2 carries no useful information and the scheme is RCI-secure, but a CCI adversary can choose one cookie that matches z and one that does not, so c_2 allows distinguishing of the chosen cookies.

- RCI $\implies$ CR: A straightforward simulation: an adversary who recovers a cookie given only ciphertexts easily distinguishes encryptions of cookies.

- CR $\not\implies$ RCI: A counterexample is constructed: an extra ciphertext component c_2 is added, consisting of a random oracle applied to the message. The adversary gets encryptions of $m'\|ck\|m''$ for m', m'' of its choice; without querying the random oracle on exactly $m'\|ck\|m''$, c_2 provides no information to the adversary, so the scheme is CR-secure. However, an RCI adversary can check the random oracle on the two given random cookies, so c_2 allows distinguishing of the given random cookies.

6.3.1 IND-CPA $\implies$ CCI: IND-CPA implies CCI

Theorem 6.3.1. Let Ψ be an IND-CPA-secure symmetric-key encryption scheme. Then Ψ is also a CCI-secure symmetric-key compression-encryption scheme for any cookie space $\mathcal{CK}$. Formally, let $\mathcal{A}$ be an adversary against the CCI security of Ψ, and let q denote the number of queries that $\mathcal{A}$ makes to its E_1 oracle. Then

$$\mathrm{Adv}^{\mathrm{CCI}}_{\Psi,\mathcal{CK}}(\mathcal{A}) \leq q \cdot \mathrm{Adv}^{\text{IND-CPA}}_{\Psi}(\mathcal{B}^{\mathcal{A}}) \ ,$$

where $\mathcal{B}$ is an algorithm, constructed using the adversary $\mathcal{A}$ as described in the proof, against the IND-CPA security of the underlying symmetric-key encryption scheme Ψ.

Proof. The proof proceeds in a sequence of games, using a hybrid approach. Each Game i proceeds as in the original CCI security experiment, except that the queries to E_1 are answered as in Figure 6.3. Let Adv^i denote the probability that game i outputs 1.

$\underline{E_1(m', m'')}$

1: **if** query $\# \leq i$ **then**
2: **return** $\Psi.\mathrm{Enc}_k(m' \| ck_0 \| m'')$
3: **else if** query $\# > i$ **then**
4: **return** $\Psi.\mathrm{Enc}_k(m' \| ck_b \| m'')$

Figure 6.3: Oracle E_1 used in Game i in proof of Theorem 6.3.1.

Game 0. This is the original CCI security game for Π. By definition,

$$\mathrm{Adv}^{\mathrm{CCI}}_{\Psi,\mathcal{CK}}(\mathcal{A}) = \mathrm{Adv}^0 \ .$$

Transition from Game $(i-1)$ to Game i, $1 \leq i \leq q$. Each hybrid transition changes how one query is answered; if the adversary's behaviour differs because of the change in answering the query, we can construct a simulator $\mathcal{B}_i$ that wins the IND-CPA game for Ψ, as shown in Figure 6.4. When the IND-CPA challenger uses $b = 0$, c^* is the encryption of $m' \| ck_{\hat{b}} \| m''$, so $\mathcal{B}_i$ is playing game $(i-1)$ with $\mathcal{A}$. When the IND-CPA challenger uses $b = 1$, c^* is the encryption of $m' \| ck_{\hat{0}} \| m''$, so $\mathcal{B}_i$ is playing game i with $\mathcal{A}$. Thus,

$$\left|\mathrm{Adv}^{i-1} - \mathrm{Adv}^i\right| \leq \mathrm{Adv}^{\text{IND-CPA}}_{\Psi}(\mathcal{B}_i^{\mathcal{A}}) \ .$$

Analysis of Game q. Since the adversary's view is independent of b in Game q, we have

$$\mathrm{Adv}^q = 0 \ .$$

$\underline{\mathcal{B}_i^{\mathcal{A},E}()}$
1: $(ck_0, ck_1, st) \xleftarrow{\$} \mathcal{A}^{E_2}()$ s.t. $|ck_0| = |ck_1|$
2: $\hat{b} \xleftarrow{\$} \{0,1\}$
3: $b' \xleftarrow{\$} \mathcal{A}^{E_1,E_2}(ck_0, ck_1, st)$
4: **return** b'

$\underline{E_2(m)}$
1: **return** $E(m)$

$\underline{E_1(m', m'')}$
1: **if** query $\# < i$ **then**
2: **return** $E(m'\|ck_0\|m'')$
3: **else if** query $\# = i$ **then**
4: Give $(m'\|ck_{\hat{b}}\|m'', m'\|ck_0\|m'')$ to IND-CPA challenger
5: Receive c^* from IND-CPA challegner
6: **return** c^*
7: **else if** query $\# > i$ **then**
8: **return** $E(m'\|ck_{\hat{b}}\|m'')$

Figure 6.4: Simulator $\mathcal{B}_i$ used in the proof of Theorem 6.3.1

Conclusion. Combining the above results, we have

$$\mathrm{Adv}^{\mathrm{CCI}}_{\Psi,\mathcal{CK}}(\mathcal{A}) \leq \sum_{i=1}^{q} \mathrm{Adv}^{\text{IND-CPA}}_{\Psi}(\mathcal{B}_i) = q \cdot \mathrm{Adv}^{\text{IND-CPA}}_{\Psi}(\mathcal{B})$$

(with a small abuse of notation in creating a single $\mathcal{B}$ from the disparate $\mathcal{B}_i$). □

6.3.2 CCI $\not\Longrightarrow$ IND-CPA: CCI does not imply IND-CPA

Theorem 6.3.2. There exists a symmetric-key compression-encryption scheme that is CCI-secure but not IND-CPA-secure.

Theorem 6.3.2 is shown using a degenerate counterexample involving the separating-secrets technique. The basic idea is that we encrypt *only* secret cookies and not the message. Technically, the CCI security definition only requires any confidentiality for the cookie portion of the ciphertext and not the rest of it, so a scheme that extracts and encrypts only the cookies is CCI-secure, but is clearly not IND-CPA-secure.

In particular, let Π be a (non-compressing) symmetric-key encryption scheme. Let $\mathcal{CK}$ be the cookie space recognized by $\llcorner$`secret[A-Za-z0-9]`$^{\lambda}\lrcorner$ and f be the corresponding filter, as described in Section 6.4.1. This filter is effective at separating out $\mathcal{CK}$.

We construct Ψ from Π and f as in Figure 6.5. We will show that Ψ is CCI-secure, but is not IND-CPA-secure.

Claim 6.3.1. Ψ in Figure 6.5 is CCI-secure, assuming Π is IND-CPA-secure.

$\underline{\Psi.\text{KeyGen}()}$

1: **return** $\Pi.\text{KeyGen}()$

$\underline{\Psi.\text{Enc}_k(m)}$

1: $pt_s \| pt_{ns} \leftarrow f(m)$
2: $c_1 \xleftarrow{\$} \Pi.\text{Enc}_k(pt_s)$
3: **return** $c_1 \| pt_{ns}$

$\underline{\Psi.\text{Dec}_k(c)}$

1: Parse $c_1 \| pt_{ns} \leftarrow c$
2: $pt_s \leftarrow \Pi.\text{Dec}_k(c_1)$
3: $m \leftarrow f^{-1}(pt_s, pt_{ns})$
4: **return** m

Figure 6.5: Scheme Ψ used in the proof of Theorem 6.3.2

Proof. Since f is effective at separating out $\mathcal{CK}$, only c_1 components carry any information about b. However, c_1 is the encryption of the secrets extracted from m' and m'' as well as ck_b. Since f is safe for $\mathcal{CK}$, the length of pt_s is the same when derived from either $m' \| ck_0 \| m''$ or $m' \| ck_1 \| m''$. Thus any adversary that can guess the bit b serves as a distinguisher for Π under chosen plaintext attack. □

Claim 6.3.2. Ψ in Figure 6.5 is not IND-CPA-secure.

Proof. The construction of a successful $\mathcal{A}$ against the IND-CPA security of Ψ is straightforward. $\mathcal{A}$ picks two distinct messages m'_0 and m'_1 that do not match the regular expression defining f, and gives these as the challenge messages to the IND-CPA challenger for Ψ. The resulting ciphertext will have an empty c_1 component (since neither m'_0 nor m'_1 has any value that will be separated out), and the second ciphertext component is unencrypted, so the adversary receives back m'_b directly. The adversary then can immediately provide a correct guess of b. □

6.3.3 CCI $\implies$ RCI: CCI implies RCI

Theorem 6.3.3. Let Ψ be a CCI-secure symmetric-key compression-encryption scheme. Then Ψ is also an RCI-secure symmetric-key compression-encryption scheme. Formally, let $\mathcal{CK}$ be a cookie space, and let $\mathcal{A}$ be an algorithm against the RCI security of Ψ. Then, for the algorithm $\mathcal{B}$ given in Figure 6.6,

$$\text{Adv}^{\text{RCI}}_{\Psi,\mathcal{CK}}(\mathcal{A}) \leq \text{Adv}^{\text{CCI}}_{\Psi,\mathcal{CK}}(\mathcal{B}^{\mathcal{A}}) \ .$$

Proof. The proof proceeds via direct simulation. We are given an adversary $\mathcal{A}$ against the RCI security of Ψ. We must construct an adversary $\mathcal{B}$ against the CCI security of Ψ; note that $\mathcal{B}$ will have access to oracles E_1 and E_2 described in the CCI security experiment. The simulator $\mathcal{B}$ is constructed in Figure 6.6.

Notice in particular that $\mathcal{B}$ uses the CCI challenger's E_1 and E_2 oracles to answer $\mathcal{A}$'s queries.

$\underline{\mathcal{B}^{\mathcal{A},E_1,E_2}()}$
1: $(ck_0, ck_1) \xleftarrow{\$} \mathcal{CK}$ s.t. $|ck_0| = |ck_1|$
2: Give (ck_0, ck_1) to the CCI challenger
3: $b' \xleftarrow{\$} \mathcal{A}^{E_1,E_2}(ck_0, ck_1, \perp)$
4: **return** b'

Figure 6.6: Simulator used in the proof of Theorem 6.3.3.

$\mathcal{B}$'s simulation of the RCI experiment to $\mathcal{A}$ is perfect. If $\mathcal{A}$'s guess b' of the b is correct in the RCI experiment, then it is also correct for the CCI experiment, and similarly when the guess is wrong. This yields the bound in the theorem. □

6.3.4 RCI $\not\Rightarrow$ CCI: RCI does not imply CCI

Theorem 6.3.4. There exists a symmetric-key compression-encryption scheme that is RCI-secure but not CCI-secure.

Theorem 6.3.4 is shown using a counterexample involving the separating-secrets technique and a point function involving a hard-coded publicly known string. The basic idea is that we append to each ciphertext a single bit representing the output of the point function on the cookie(s) in the message. Since randomly chosen cookies are highly unlikely to be the same as the hard-coded value in the point function, this extra bit provides no useful information for an RCI adversary, but a chosen-cookie adversary could easily pick one cookie to be the hard-coded value and one not to be, the bit thereby allowing him to easily distinguish the two.

In particular, let Π be a symmetric-key encryption scheme and let Γ be a compression scheme. Let $\mathcal{CK}$ be the cookie space recognized by `␣secret[A-Za-z0-9]`$^{\lambda}$`␣` and f be the corresponding filter, as described in Section 6.4.1. Recall this filter is safe for $\mathcal{CK}$.

Let $\Psi = \Pi \circ \mathrm{SS}_{f,\Gamma}$ be the symmetric-key compression-encryption scheme constructed using the separating-secrets technique. We will show in Theorem 6.4.1 that Ψ is CCI-secure if Π is IND-CPA secure. By Theorem 6.3.3, Ψ is thus also RCI-secure.

Additionally, let $z \in \mathcal{CK}$, and define the point function

$$g_z(x) = \begin{cases} 1, & \text{if } z \preceq x, \\ 0, & \text{otherwise.} \end{cases}$$

We will construct Ψ' as in Figure 6.7 from $\Psi = \Pi \circ \mathrm{SS}_{f,\Gamma}$ and g_z. We will show that Ψ' remains RCI-secure, but is not CCI-secure.

$\underline{\Psi'.\mathrm{KeyGen}()}$
1: **return** $\Psi.\mathrm{KeyGen}()$

$\underline{\Psi'.\mathrm{Enc}_k(m)}$
1: $pt_s \| \widetilde{pt_{ns}} \leftarrow \mathrm{SS}_{f,\Gamma}.\mathrm{Comp}(m)$
2: $c_1 \xleftarrow{\$} \Pi.\mathrm{Enc}_k(pt_s \| \widetilde{pt_{ns}})$
3: $c_2 \leftarrow g_z(pt_s)$
4: **return** $c_1 \| c_2$

$\underline{\Psi'.\mathrm{Dec}_k(c)}$
1: Parse $c_1 \| c_2 \leftarrow c$
2: **return** $\Psi.\mathrm{Dec}_k(c_1)$

Figure 6.7: Scheme Ψ' used in the proof of Theorem 6.3.4

Claim 6.3.3. Ψ' in Figure 6.7 is RCI-secure, assuming Π is IND-CPA-secure and $\mathcal{CK}$ is large.

Proof. Let ck_0^* and ck_1^* denote the random cookies to be distinguished.

Because $\Pi \circ \mathrm{SS}_{f,\Gamma}$ is RCI-secure, c_1 gives the adversary no advantage in guessing the bit b. We need to assess whether c_2 helps the adversary at all in winning the RCI game.

The second ciphertext component c_2 is only useful to the adversary if the adversary can construct a pair (m', m'') such that c_2 is different for $m' \| ck_0^* \| m''$ versus $m' \| ck_1^* \| m''$.

Consider the construction of pt_s from m in $\Psi'.\mathrm{Enc}_k(m)$. Since pt_s consists of a comma-separated list of cookies, and no cookie contains a comma, $g_z(pt_s) = 1$ if and only if there exists some i such that $g_z(ck_i) = 1$, where m is parsed as $m_0 \| ck_1 \| m_1 \| ck_2 \| m_2 \| \ldots \| ck_n \| m_n$.

Now consider the handling of $m = m' \| ck_b^* \| m''$ for m', m'' of the adversary's choosing. By the argument above, $g_z(pt_s) = g_z(ck_b^*) \vee g_z(m') \vee g_z(m'')$. Moreover, $g_z(ck_0^*) \neq g_z(ck_1^*)$ if and only if one of them is equal to z but the other is not. Since ck_0^* and ck_1^* are chosen uniformly at random from $\mathcal{CK}$, this occurs with probability at most $2/|\mathcal{CK}|$. □

Claim 6.3.4. Ψ' in Figure 6.7 is not CCI-secure.

Proof. The construction of a successful $\mathcal{A}$ against the CCI security of Ψ' is straightforward. Note that $\mathcal{A}$ knows the value z in the point function g_z. Thus, $\mathcal{A}$ could issue a CCI challenge with $ck_0 = z$ and $ck_1 \neq z$, and then make the query $E_1(\epsilon, \epsilon)$, thereby obtaining $c_1 = \mathrm{Enc}_k(ck_b)$ and $c_2 = g_z(ck_b)$. The last bit of such a ciphertext immediately indicates whether $ck_b = z$ or not. □

6.3.5 RCI $\implies$ CR: RCI implies CR

Theorem 6.3.5. Let Ψ be an RCI-secure symmetric-key compression-encryption scheme. Then Ψ is also a CR-secure symmetric-key compression-encryption scheme. Formally, let $\mathcal{CK}$ be a cookie space, and let $\mathcal{A}$ be an algorithm against the CR security of Ψ. Then, for the algorithm $\mathcal{B}$ given in Figure 6.8,

$$\mathrm{Adv}^{\mathrm{CR}}_{\Psi,\mathcal{CK}}(\mathcal{A}) \leq \mathrm{Adv}^{\mathrm{RCI}}_{\Psi,\mathcal{CK}}(\mathcal{B}^{\mathcal{A}}) \ .$$

Proof. The proof proceeds via direct simulation. We are given an adversary $\mathcal{A}$ against the CR security of Ψ. We must construct an adversary $\mathcal{B}$ against the RCI security of Ψ; note that $\mathcal{B}$ will have access to oracles E_1 and E_2 described in the RCI security experiment. The simulator $\mathcal{B}$ is constructed in Figure 6.8. Notice in particular that $\mathcal{B}$ uses the RCI challenger's E_1 and E_2 oracles to answer $\mathcal{A}$'s queries.

$\underline{\mathcal{B}^{\mathcal{A},E_1,E_2}(ck_0, ck_1)}$

1: $ck' \xleftarrow{\$} \mathcal{A}^{E_1,E_2}()$
2: **if** $ck' = ck_0$ **then**
3: **return** 0
4: **else if** $ck' = ck_1$ **then**
5: **return** 1
6: **else**
7: **return** $b' \xleftarrow{\$} \{0,1\}$

Figure 6.8: Simulator used in the proof of Theorem 6.3.5.

$\mathcal{B}$'s simulation of the CR experiment to $\mathcal{A}$ is perfect: the value ck_b used by the E_1 oracle in the RCI challenger was indeed chosen at random, just as in the CR experiment, and is indeed consistent.

If $\mathcal{A}$'s guess ck' of the cookie is correct in the CR simulation, then it is also correct for the RCI experiment, and so $\mathcal{B}$'s output will be correct. If $\mathcal{A}$'s guess is wrong, then $\mathcal{B}$ does as good as random guessing. □

6.3.6 CR $\not\Rightarrow$ RCI: CR does not imply RCI

Theorem 6.3.6. There exists a symmetric-key compression-encryption scheme that is CR-secure but not RCI-secure.

Theorem 6.3.6 is shown using a counterexample involving a random oracle. The basic idea is that we append to each ciphertext the output of the random oracle applied to the message. For an adversary who is trying to guess an unknown cookie, this provides no information unless it queries the random oracle on the cookie itself, but a random-cookie adversary, who knows that the cookie is one of two values, could easily determine which by querying both to the random oracle.

Let Ψ be a symmetric-key compression-encryption scheme. Let $H : \{0,1\}^* \to \{0,1\}^\lambda$ be a random oracle. Construct Ψ' as shown in Figure 6.9.

$\underline{\Psi'.\mathrm{KeyGen}()}$
1: **return** $\Psi.\mathrm{KeyGen}()$

$\underline{\Psi'.\mathrm{Enc}_k(m)}$
1: $c_1 \xleftarrow{\$} \Psi.\mathrm{Enc}_k(m)$
2: $c_2 \leftarrow H(m)$
3: **return** $c_1 \| c_2$

$\underline{\Psi'.\mathrm{Dec}_k(c)}$
1: Parse $c_1 \| c_2 \leftarrow c$
2: **return** $\Psi.\mathrm{Dec}_k(c_1)$

Figure 6.9: Scheme Ψ' used in the proof of Theorem 6.3.6

Claim 6.3.5. Ψ' is CR-secure in the random oracle model, assuming Ψ is CR-secure and $\mathcal{CK}$ is sufficiently large. Formally,

$$\mathrm{Adv}^{\mathrm{CR}}_{\Psi',\mathcal{CK}}(\mathcal{A}) \leq q_H \cdot \ell \cdot \mathrm{Adv}^{\mathrm{CR}}_{\Psi,\mathcal{CK}}(\mathcal{B}^{\mathcal{A}}) \ ,$$

where q_H is the number of queries that $\mathcal{A}$ makes to the random oracle, ℓ is the maximum length of a message queried by $\mathcal{A}$ to the random oracle, and $\mathcal{B}$ is the algorithm given in Figure 6.10.

Proof. Let ck^* denote the cookie to be recovered.

The intuition of the proof is as follows. Because Ψ is CR-secure, c_1 does not help the adversary in guessing the cookie ck^*. We need to assess whether c_2 helps the adversary at all in winning the CR game. The second ciphertext component c_2 is only useful to the adversary if the adversary queries the random oracle on the plaintext of c_1, which would mean that the adversary queries the random oracle on the target cookie ck^*. We can thus use this to win the CR experiment for Ψ'.

More precisely, the proof proceeds via direct simulation. We are given an adversary $\mathcal{A}$ against the CR security of Ψ'. We must construct an adversary $\mathcal{B}$ against the CR security of Ψ; note that $\mathcal{B}$ will have access to oracles E_1 and E_2 described in the CR security experiment for for Ψ. The simulator $\mathcal{B}$ is constructed in Figure 6.10.

$\underline{\mathcal{B}^{\mathcal{A},E_1,E_2}()}$

1: $i \xleftarrow{\$} \{1,\ldots,q_H\}$
2: $j \xleftarrow{\$} \{1,\ldots,\ell\}$
3: $rand \xleftarrow{\$} \mathcal{CK}$
4: $ck' \xleftarrow{\$} \mathcal{A}^{E_1',E_2',H}()$

$\underline{E_1'(m',m'')}$

1: $c_1 \xleftarrow{\$} E_1(m',m'')$
2: $c_2 \leftarrow H(m'\|rand\|m'')$
3: **return** $c_1\|c_2$

$\underline{E_2'(m)}$

1: $c_1 \xleftarrow{\$} E_2(m)$
2: $c_2 \leftarrow H(m)$
3: **return** $c_1\|c_2$

$\underline{H(m)}$

1: **if** query $\#= i$ **then**
2: Parse m to identify every substring of m that is in $\mathcal{CK}$
3: Pick one uniformly at random
4: Output it to the CR challenger for Ψ
5: **else**
6: Answer $H(m)$ as normal for a random oracle

Figure 6.10: Simulator used in the proof of Theorem 6.3.6.

$\mathcal{B}$'s simulation of the CR experiment for Ψ' to $\mathcal{A}$ is perfect so long as E_1' and H remain consistent. The only time an inconsistency arises is if $\mathcal{A}$ queries H on $m'\|ck^*\|m''$ for some m', m'' that it also queries to E_1'. With probability $1/q$, $\mathcal{B}$ will correctly guess that the first such query to H is the ith query. Moreover, with probability at least $1/\ell$, $\mathcal{B}$ will correctly guess which substring of that query to H is the target cookie c^*. For simplicity, we ignore the possibility of collisions on the output of the random oracle. □

Claim 6.3.6. Ψ' is not RCI-secure.

Proof. The construction of a successful $\mathcal{A}$ against the RCI security of Ψ' is straightforward. An adversary $\mathcal{A}$ against the RCI security of Ψ' is told that the target cookie is one of two values, ck_0 and ck_1. $\mathcal{A}$ could make the query $E_1(\epsilon,\epsilon)$, thereby obtaining $c_1 = \mathrm{Enc}_k(ck_b)$ and $c_2 = H(ck_b)$. $\mathcal{A}$ could then query $H(ck_0)$ and $H(ck_1)$; one of these will equal c_2, telling the adversary the value of b. □

6.4 Technique 1: Separating Secrets from User Inputs

In this section we analyze a mitigation technique against attacks that recover secrets from compressed data: separating secrets from user inputs. The basic idea of separating secrets from user inputs is: given an input, use a filter to separate all the secrets from the rest of the content, including user inputs. Then the rest of the content is compressed, while the secrets are kept uncompressed. This mitigation technique is a generic mitigation technique against a whole class of compression-based side-channel attacks.

6.4.1 The Scheme

Definition 6.4.1 (Filter)**.** A *filter* is an invertible (efficient) function $f : \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$.

Given a filter f and a compression scheme Γ, the separating-secrets scheme $\mathrm{SS}_{f,\Gamma}$ is given in Figure 6.11.

$\underline{\mathrm{SS}_{f,\Gamma}.\mathrm{Comp}(m)}$

1: $(pt_s, pt_{ns}) \leftarrow f(m)$
2: $\widetilde{pt_{ns}} \leftarrow \Gamma.\mathrm{Comp}(pt_{ns})$
3: **return** $pt_s \| \widetilde{pt_{ns}}$

$\underline{\mathrm{SS}_{f,\Gamma}.\mathrm{Decomp}(pt)}$

1: Parse $pt_s \| \widetilde{pt_{ns}} \leftarrow pt$
2: $pt_{ns} \leftarrow \Gamma.\mathrm{Decomp}(\widetilde{pt_{ns}})$
3: $m \leftarrow f^{-1}(pt_s, pt_{ns})$
4: **return** m

Figure 6.11: Abstract separating-secrets compression scheme SS

Our results will make use of the following two conditions on filters. Intuitively, a filter is *effective* if it removes cookies from an input string, and is *safe* if no prefix/suffix can fool the filter into separating out one cookie but not another.

Definition 6.4.2 (Effective filter)**.** Let $\mathcal{CK}$ be a cookie space, and let f be a filter. We say that f is *effective at separating out* $\mathcal{CK}$ if, for all $ck \in \mathcal{CK}$ and all m', m'', we have that $ck \not\preceq y$, where $(x, y) = f(m' \| ck \| m'')$.

Definition 6.4.3 (Safe filter)**.** Let $\mathcal{CK}$ be a cookie space, and let f be a filter. We say that f is *safe for* $\mathcal{CK}$ if, for all $ck_0, ck_1 \in \mathcal{CK}$ such that $|ck_0| = |ck_1|$ and all m', m'', we have that $|x_0| = |x_1|$ and $y_0 = y_1$, where $(x_0, y_0) = f(m' \| ck_0 \| m'')$ and $(x_1, y_1) = f(m' \| ck_1 \| m'')$.

Example cookie space and filter. Let $\lambda \in \mathbb{N}$ and let $\mathcal{CK}$ be the set of alphanumeric strings starting with the literal "`secret`" and starting and ending with a space (denoted by $\llcorner$), i.e., strings matched by the regular expression

$$\llcorner\texttt{secret[A-Za-z0-9]}^{\lambda}\llcorner$$

Let f be a filter that uses the above regular expression to separate out secrets. Consider a string of the form $m = m_0 \llcorner ck_1 \llcorner m_1 \llcorner ck_2 \llcorner m_2 \llcorner \ldots \llcorner ck_n \llcorner m_n$, where m_i contains no substring matching the above regular expression and ck_i is a string completely matching the above regular expression (excluding the initial and terminal space $\llcorner$). Then $f(m) = (pt_s, pt_{ns})$, where $pt_s = ck_1 \| \ldots \| ck_n$ and $pt_{ns} = m_0 \| \tau \| m_1 \| \tau \| \ldots \| m_n$, and τ represents a fixed replacement token that can not appear as a substring of any $m \in \mathcal{M}$. The above filter f is effective at separating out and safe for the above $\mathcal{CK}$. The intuitive reason for this is that, since each cookie begins and ends with a character $\llcorner$ which does not appear within the cookie, no prefix or suffix can cause the filter to not separate a cookie.

Claim 6.4.1. The above filter f is effective at separating out and safe for the above $\mathcal{CK}$.

Proof sketch. Since each cookie begins and ends with a character $\llcorner$ which does not appear within the cookie, no prefix or suffix can cause the filter to not separate a cookie.

More precisely, for any $\llcorner ck \llcorner \in \mathcal{CK}$ and any m' that contains no substring matching the above regular expression and any $m'' \neq \epsilon$, we have that $f(m' \| \llcorner ck \llcorner \| m'') = (ck \| x,\ m' \| \tau \| y)$, where $(x, y) = f(m'')$. Such an f is effective at separating out $\mathcal{CK}$ since it separates every substring of m that is a cookie into the first component of the output. Moreover, f is safe for $\mathcal{CK}$ by recursively applying the above identity. □

6.4.2 CCI Security of Basic Separating-Secrets Technique

In this section we analyze the security of separating-secrets mitigation technique according to the CCI notion. Let $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec})$ be an IND-CPA-secure symmetric-key encryption scheme and $\text{SS}_{f,\Gamma}$ be the separating-secrets compression scheme given in Figure 6.11. We consider the security of the resulting symmetric-key compression-encryption scheme $\Pi \circ \text{SS}_{f,\Gamma}$, showing that, if the filter f safely

separates out cookies, then breaking chosen cookie indistinguishability of $\Pi \circ \mathrm{SS}_{f,\Gamma}$ is as hard as breaking indistinguishability (IND-CPA) of encryption scheme Π.

Theorem 6.4.1. Let Π be a symmetric-key encryption scheme and let Γ be a compression scheme. Let $\mathcal{CK}$ be a cookie space, and let f be a filter that is safe for $\mathcal{CK}$. Let $\mathcal{A}$ be any adversary against the CCI security of the separating-secrets symmetric-key compression-encryption scheme $\Pi \circ \mathrm{SS}_{f,\Gamma}$, and let q denote the number of queries that $\mathcal{A}$ makes to its E_1 oracle. Then $\mathrm{Adv}^{\mathrm{CCI}}_{\Pi \circ \mathrm{SS}_{f,\Gamma},\mathcal{CK}}(\mathcal{A}) \leq q \cdot \mathrm{Adv}^{\text{IND-CPA}}_{\Pi}(\mathcal{B}^{\mathcal{A}})$, where $\mathcal{B}$ is an algorithm, constructed using the adversary $\mathcal{A}$ as described in the proof, against the IND-CPA security of the symmetric-key encryption scheme Π.

Proof. The proof proceeds in a sequence of games, using a hybrid approach. Each Game i proceeds as in the original CCI security experiment, except that the queries to E_1 are answered as in Figure 6.12. Let Adv^i denote the probability that game i outputs 1.

$\underline{E_1(m', m'')}$
1: **if** query $\# \leq i$ **then**
2: **return** $\Pi.\mathrm{Enc}_k(\mathrm{SS}_{f,\Gamma}(m' \| ck_0 \| m''))$
3: **else if** query $\# > i$ **then**
4: **return** $\Pi.\mathrm{Enc}_k(\mathrm{SS}_{f,\Gamma}(m' \| ck_b \| m''))$

Figure 6.12: Oracle E_1 used in Game i in proof of Theorem 6.4.1.

Game 0. This is the original CCI security game for Π. By definition,

$$\mathrm{Adv}^{\mathrm{CCI}}_{\Pi \circ \mathrm{SS}_{f,\Gamma},\mathcal{CK}}(\mathcal{A}) = \mathrm{Adv}^0 \ .$$

Transition from Game $(i-1)$ to Game i, $1 \leq i \leq q$. Each hybrid transition changes how one query is answered; if the adversary's behaviour differs because of the change in answering the query, we can construct a simulator $\mathcal{B}_i$ that wins the IND-CPA game for Ψ, as shown in Figure 6.13. When the IND-CPA challenger uses $b = 0$, c^* is the encryption of the separating-secrets compression of $m' \| ck_{\hat{b}} \| m''$, so $\mathcal{B}_i$ is playing game $(i-1)$ with $\mathcal{A}$. When the IND-CPA challenger uses $b = 1$, c^* is the encryption of the separating-secrets compression of $m' \| ck_0 \| m''$, so $\mathcal{B}_i$ is playing game i with $\mathcal{A}$. Since f is safe for $\mathcal{CK}$, the separating-secrets compressions of $m' \| ck_0 \| m''$ and $m' \| ck_1 \| m''$ have the same length, and thus the pair of chosen

$\underline{\mathcal{B}_i^{\mathcal{A},E}()}$

1: $(ck_0, ck_1, st) \xleftarrow{\$} \mathcal{A}^{E_2}()$ s.t. $|ck_0| = |ck_1|$
2: $\hat{b} \xleftarrow{\$} \{0,1\}$
3: $b' \xleftarrow{\$} \mathcal{A}^{E_1,E_2}(ck_0, ck_1, st)$
4: **return** b'

$\underline{E_2(m)}$

1: **return** $\Pi.\mathrm{Enc}_k(SS_{f,\Gamma}.\mathrm{Comp}(m))$

$\underline{E_1(m', m'')}$

1: **if** query $\# < i$ **then**
2: **return** $\Pi.\mathrm{Enc}_k(SS_{f,\Gamma}.\mathrm{Comp}(m'\|ck_0\|m''))$
3: **else if** query $\# = i$ **then**
4: $pt\|\widetilde{pt_{ns}} \leftarrow SS_{f,\Gamma}.\mathrm{Comp}(m'\|ck_{\hat{b}}\|m'')$
5: $pt'\|\widetilde{pt_{ns}}' \leftarrow SS_{f,\Gamma}.\mathrm{Comp}(m'\|ck_0\|m'')$
6: Give $(pt\|\widetilde{pt_{ns}}, pt'\|\widetilde{pt_{ns}}')$ to IND-CPA challenger
7: Receive c^* from IND-CPA challenger
8: **return** c^*
9: **else if** query $\# > i$ **then**
10: **return** $\Pi.\mathrm{Enc}_k(SS_{f,\Gamma}.\mathrm{Comp}(m'\|ck_{\hat{b}}\|m''))$

Figure 6.13: Simulator $\mathcal{B}_i$ used in the proof of Theorem 6.4.1

messages given from the simulator in E_1 to the IND-CPA challenger is valid according to the IND-CPA experiment. Thus,

$$\left|\mathrm{Adv}^{i-1} - \mathrm{Adv}^{i}\right| \leq \mathrm{Adv}_{\Psi}^{\text{IND-CPA}}(\mathcal{B}_i^{\mathcal{A}}) \ .$$

Analysis of Game q. Since the adversary's view is independent of b in Game q, we have

$$\mathrm{Adv}^{q} = 0 \ .$$

Conclusion. Combining the above results, we have

$$\mathrm{Adv}_{\Psi,\mathcal{CC}}^{\text{CCI}}(\mathcal{A}) \leq \sum_{i=1}^{q} \mathrm{Adv}_{\Psi}^{\text{IND-CPA}}(\mathcal{B}_i^{\mathcal{A}}) = q \cdot \mathrm{Adv}_{\Psi}^{\text{IND-CPA}}(\mathcal{B}^{\mathcal{A}})$$

(with a small abuse of notation in creating a single $\mathcal{B}$ from the disparate $\mathcal{B}_i$). □

6.4.3 Separating Secrets in HTML

Separating secrets from user inputs is a realistic mitigation technique against the BREACH attack: in the application layer, some fields which contain secrets (such as anti-CSRF tokens) can be identified and separated from the HTTP response body. In order to implement separating secrets from user inputs in HTML we need to describe a filter f_{HTML}.

One possible method to separate secrets in HTML is to separate the content assigned to the `value` attribute of HTML elements. Among other uses, the `value` attribute defines the value of a specific field in a form. The HTML code segment of Figure 6.14 shows inclusion of a secret anti-CSRF token as a hidden `input` field in a web form, which will appear in a HTML response body. By separating the content in the `value` attribute, we separate the anti-CSRF token.

```
<form action="/money_transfer" method="post">
<input type="hidden" name="csrftoken"
       value="OWT4NmQlODE4ODRjN2Q1NTlhMmZlYWE...">
...
</form>
```

Figure 6.14: HTML code segment showing inclusion of anti-CSRF token in a web form

The following (case-insensitive) regular expression can be used to separate out quoted anti-CSRF tokens in the `value` attribute of HTML elements:

```
value\s*=\s*"[A-Za-z0-9]+"|value\s*=\s*'[A-Za-z0-9]+'
```

This filter is effective at separating out and safe for the implied set of cookies, in the sense of Definitions 6.4.2 and 6.4.3.

However, the above regular expression is not perfect, highlighting the challenges of using heuristic techniques to separate out secrets.

First, the above regular expression will also capture the `value` attribute of HTML elements other than hidden `input` elements, such as `option`, which may not need to be treated as secret, so it is not as efficient as it could be.

Second, the above regular expression does not capture anti-CSRF tokens in unquoted `value` attributes, such as `value=OWT4NmQl`, which are allowed by the HTML specification. While it is easy to add an additional term such as `|value\s*=\s*[A-Za-z0-9]+` to the regular expression to capture unquoted attributes, this filter would no longer be effective in the sense of Definition 6.4.2: if a cookie is `value=OWT4NmQl`, and the adversary constructs $m' =$ `value=`, then $m'\|ck =$ `value=value=OWT4NmQl`, and the filter applied to $m'\|ck$ would separate out `value=value` as the cookie and leave `=OWT4NmQl` unprotected.

Website	Uncompressed	gzip full page		Separating secrets		Fixed dictionary	
Google.com	145 599	41 455	(3.51×)	41 502	(3.51×)	117 794	(1.23×)
Facebook.com	48 226	13 785	(3.50×)	15 863	(3.04×)	35 036	(1.37×)
Youtube.com	467 928	41 813	(11.19×)	41 893	(11.17×)	181 676	(2.58×)
Yahoo.com	444 408	82 572	(5.38×)	83 342	(5.33×)	318 386	(1.40×)
Baidu.com	74 979	17 519	(4.28×)	17 727	(4.23×)	55 950	(1.34×)
Wikipedia.org	48 548	11 217	(4.33×)	11 809	(4.11×)	38 406	(1.26×)
Twitter.com	57 777	12 520	(4.61×)	16 618	(3.48×)	39 712	(1.46×)
Qq.com	626 297	124 108	(5.05×)	125 747	(4.98×)	519 830	(1.21×)
Amazon.com	234 609	54 922	(4.27×)	56 278	(4.17×)	150 924	(1.55×)
Taobao.com	192 068	23 658	(8.12×)	23 898	(8.04×)	93 410	(2.06×)

Table 6.1: Compression performance (file size in bytes and compression ratio) for separating secrets (Section 6.4) and fixed dictionary (Section 6.5) techniques

6.4.4 Results on Separating-Secrets in HTML

Table 6.1 shows the result of applying the above regular expression to separate secrets on the top 10 global websites of Alexa Top Sites. As most pages contain little data in `value` attributes, the total amount of space required to transmit the separated secrets plus the remaining data is not much more than when the full page is compressed. (Table 6.1 also contains performance results of the fixed dictionary technique, to be discussed in Section 6.5.)

Source code of separating-secrets from HTML files and separating-secrets experiment are in Appendix A.3 and A.4 respectively.

6.4.5 Discussion

The main drawback of the separating secrets technique is that the separation filter must be application-dependent. We noted already the challenges in using the heuristic regular expression above to capture anti-CSRF tokens: it may separate out non-secrets as well as secrets (which yields suboptimal compression) and it does not capture unquoted tokens (which is a problem for security).

Moreover, this HTML filter also only captures secrets in a `value` attribute, which does not necessarily capture all values that might be considered sensitive. For example, should the titles of books in a search results page on an shopping site be considered secret? If so, an alternative separation filter would have to be developed. To provide complete certainty, secret separation would require additional markup with which the developer clearly identifies which data should be treated as secret. Otherwise, any sensitive values which are not separated may be compressed together with user inputs and other application data, and hence remain open to the compression-based side-channel.

6.5 Technique 2: Fixed-Dictionary Compression

The CRIME and BREACH attacks work because the dictionary constructed by the DEFLATE compression algorithm is adaptive: if the attacker injects a substring of the target secret into the plaintext nearby the secret itself, then the plaintext will compress more because of the repeated substring. Some early compression algorithms were non-adaptive, using a fixed dictionary mechanism. For example, Pike [Pik80] used a fixed dictionary of 205 popular English words and a variable length coding mechanism to compress typical English text at a rate of less than 4 bits per character. Another recent algorithm, Smaz [San09], similarly uses a fixed dictionary consisting of common digrams and trigrams from English and HTML source code, allowing it to compress even very short strings. Because the CRIME and BREACH attacks rely on the adaptivity of the compression dictionary, fixed-dictionary algorithms can offer resistance to such attacks while still providing some compression, albeit not as good as adaptive compression.

In this section, we investigate the use of fixed-dictionary compression in the context of encryption. We describe the basic idea of fixed-dictionary compression. We show that fixed-dictionary compression-encryption schemes can satisfy cookie recovery security for sufficiently large cookies. We then present an example of a modern fixed-dictionary compression algorithm and report on the compression ratios achieved by our algorithm.

6.5.1 The Scheme

In general, fixed-dictionary compression schemes work by advancing through the string x and looking to see if the current substring appears in the dictionary $\mathcal{D}$: if it does, then an encoding of the index of the substring is recorded, otherwise an encoding of the current substring is recorded. The compression scheme must specify the encoding rules in a way that unambiguously discriminates between the two cases to allow for correct decompression.

An abstract version of a fixed-dictionary fixed-width compression algorithm FD is given in Figure 6.15. FD checks if the current substring of length w appears in the dictionary $\mathcal{D}$. If it does, it records the index of the substring in $\mathcal{D}$ and advances w characters. If it does not, it records the next ℓ characters directly, then advances. (Using $\ell > 1$ but $\ell < w$ may be more efficient when it comes to encodings.) One could treat $\mathcal{D}$ either as a set of strings (recording which element

is matched) or a long string (recording the starting and ending position of the matching substring); we will use the latter in the rest of this section.

$\underline{\text{FD}_{\mathcal{D},w,\ell}.\text{Comp}(x)}$

```
1: y ← ϵ
2: i ← 1
3: while i ≤ |x| − w + 1 do
4:     if x_{i:w} ⪯ D then
5:         y ← y ‖ encoding of ind_D(x_{i:w})
6:         i ← i + w
7:     else
8:         y ← y ‖ encoding of x_{i:ℓ}
9:         i ← i + ℓ
10: return y
```

$\underline{\text{FD}_{\mathcal{D},w,\ell}.\text{Decomp}(y)}$

```
1: x ← ϵ
2: i ← 1
3: while i ≤ |y| do
4:     if y_i encodes an index then
5:         x ← x ‖ D_{y_i:w}
6:         i ← i + 1
7:     else
8:         x ← x ‖ decoding of y_{i:ℓ'}
9:         i ← i + ℓ'
10: return x
```

Figure 6.15: Abstract fixed-dictionary fixed-width compression scheme FD Note the simplification that ℓ characters of x are encoded as ℓ' characters of y.

For example, if $\mathcal{D}$ =“`cookierecoveryattack`”, then $\text{FD}_{\mathcal{D},4,2}$.Comp(“`recover the cookie`”) yields `7ver_the_1ie`.

6.5.2 CR Security of Basic Fixed-Dictionary Technique

Let Π be a symmetric-key encryption scheme. Let $\mathcal{D}$ be a dictionary of length d and $\text{FD}_{\mathcal{D},w,\ell}$ be the abstract fixed-dictionary compression scheme in Figure 6.15.

Suppose the cookie space is binary strings of length 8λ, or equivalently byte strings of length λ: $\mathcal{CK} = \{\texttt{0x00}, \ldots, \texttt{0xFF}\}^{\lambda}$.

If Π is a secure encryption scheme, then, intuitively, the only way the adversary can learn information about the cookie from seeing ciphertexts $\text{Enc}_k(\cdot\|ck\|\cdot)$ and $\text{Enc}_k(\cdot)$ is from the length of the ciphertext: if some substring of ck appears in the dictionary $\mathcal{D}$, then ck will compress, and that length difference tells the adversary that the secret cookie is restricted to some subset of $\mathcal{CK}$ matching $\mathcal{D}$.

The situation is subtler in the full CR experiment: the attacker can provide m' and m'' and get $\text{Enc}_k(\text{Comp}(m'\|ck\|m''))$. If the last few bytes of m' followed by the first few bytes of ck appear in $\mathcal{D}$, then the string will compress more. This allows the attacker to carry out a CRIME-like attack on the first few bytes of ck.

For example, let $w = 4$ and suppose $\mathcal{D}$ = `1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ` and $\mathcal{CK} = [\texttt{0-9A-F}]^{\lambda}$. The attacker can query m' = `890`, m' = `90A`, m' = `0AB`, ... In exactly one case, the adversary's m' combined with the cookie's first byte will be in the dictionary, telling the adversary ck_1. For example, if

$ck_1 = \texttt{B}$, then when the adversary queries $m' = \texttt{90A}$, the value that is compressed and then encrypted is $m' \| ck \| m'' = \texttt{90AB}\ldots$, which is a substring of $\mathcal{D}$.

While this allows the attacker to recover the first byte or two of the secret cookie with decent probability, it drops off exponentially; a similar argument applies to the last few bytes of the secret cookie. Theorem 6.5.1 captures this issue. Theorem 6.5.1 only provides quantifiable security when the cookie length n is significantly bigger than the compression window w. Additionally, this type of attack on the first/last few bytes of the cookie precludes *indistinguishable* security, which is why we focus on cookie *recovery* here. (Admittedly, in some settings recovering the first/last few cookie bytes may still be quite damaging.)

Theorem 6.5.1. Let Π be a symmetric-key encryption scheme. Let $\mathcal{D}$ be a dictionary of d words, each of length ℓ. Let w be a positive integer. Let $\mathcal{CK} = \Omega^n$. Let $\mathcal{A}$ be any adversary against the cookie recovery security of the fixed-dictionary symmetric-key compression-encryption scheme $\Pi \circ \mathrm{FD}_{\mathcal{D},w,\ell}$. Then $\mathrm{Adv}^{\mathrm{CR}}_{\Pi \circ \mathrm{FD}_{\mathcal{D},w,\ell}}(\mathcal{A}) \leq \mathrm{Adv}^{\text{IND-CPA}}_{\Pi}(\mathcal{B}) + 2^{-\Delta}$, where $\mathcal{B}$ is an algorithm, constructed using adversary $\mathcal{A}$, against the IND-CPA security of the symmetric-key encryption scheme Π, and

$$\Delta \geq \left(1 - d\left(1 - \left(1 - \frac{1}{|\Omega|^w}\right)^{n-3w+1}\right)\right) \cdot \log_2\left(|\Omega|^{n-2w} - |\Omega|^{n-2w} \cdot d\left(1 - \left(1 - \frac{1}{|\Omega|^w}\right)^{n-3w+1}\right)\right) .$$

For example, for cookies of $n = 16$ bytes, with a dictionary of $d = 4000$ words each of length $w = 4$ bytes, we have $\Delta \geq 63.999695$. Doubling d gives $\Delta \geq 63.999391$.

Proof.

Probability bounds, no prefix/suffix. In this section, we compute the amount of information given to the adversary from knowing the length of the compressed cookie, without any adversarially chosen prefix or suffix. This can be computed by calculating the amount of information given by knowing how many substrings of the cookie appear in the dictionary. For the analysis, we treat $\mathcal{D}$ as a set of strings.

First we calculate the probability that a given string is a substring of a randomly chosen cookie.

Lemma 6.5.1. Let $x \in \Omega^w$ be a word, and let $ck \xleftarrow{\$} \Omega^n = \mathcal{CK}$ be a random string of n characters. Then $\Pr(x \preceq ck) \leq 1 - \left(1 - \frac{1}{|\Omega|^w}\right)^{n-w+1}$.

Proof.

$$\begin{aligned}
\Pr(x \preceq ck) &= 1 - \Pr(x \not\preceq ck) \\
&= 1 - \Pr((x \neq ck_{1:w}) \wedge (x \neq ck_{2:w}) \wedge \cdots \wedge (x \neq ck_{n-w+1:w})) \\
&\leq 1 - \Pr(x \neq ck_{1:w}) \Pr(x \neq ck_{2:w}) \ldots \Pr(x \neq ck_{n-w+1:w}) \\
&= 1 - \left(1 - \frac{1}{|\Omega|^w}\right)^{n-w+1}
\end{aligned}$$

□

We now compute that probability that one of a set of given strings is a substring of a randomly chosen cookie:

Lemma 6.5.2. Let $\mathcal{D} \subseteq \Omega^w$ with $|\mathcal{D}| = d$ be a dictionary of d words of w characters. Let $ck \xleftarrow{\$} \Omega^n = \mathcal{CK}$ be a random string of n characters. Then

$$\Pr(\exists x \in \mathcal{D} : x \preceq ck) \leq d\left(1 - \left(1 - \frac{1}{|\Omega|^w}\right)^{n-w+1}\right) .$$

Proof. Suppose $\mathcal{D} = \{x_1, x_2, \ldots, x_d\}$.

$$\begin{aligned}
\Pr(\exists x \in \mathcal{D} : x \preceq ck) &= \Pr((x_1 \preceq ck) \vee (x_2 \preceq ck) \vee \cdots \vee (x_d \preceq ck)) \\
&\leq \sum_{i=1}^{d} \Pr(x_i \preceq ck) \\
&\leq d\left(1 - \left(1 - \frac{1}{|\Omega|^w}\right)^{n-w+1}\right)
\end{aligned}$$

□

Recall the definition of conditional entropy for random variables X and Y:

$$\begin{aligned}
H(Y \mid X) &= \sum_{x \in \mathrm{supp}(X)} \Pr(X = x) H(Y \mid X = x) \\
&= -\sum_{x \in \mathrm{supp}(X)} \Pr(X = x) \\
&\quad \cdot \sum_{y \in \mathrm{supp}(Y)} \Pr(Y = y \mid X = x) \log_2 \Pr(Y = y \mid X = x) .
\end{aligned}$$

We now compute the amount of entropy about the cookie given knowledge about the number of substrings of the cookie that appear in the dictionary:

Lemma 6.5.3. Fix $\mathcal{D}$. Let $\#\mathrm{SUB}(ck)$ denote the number of substrings of ck that appear in $\mathcal{D}$. Suppose CK is a uniform random variable on $\mathcal{CK}$. Then

$$H(CK \mid \#\mathrm{SUB}(CK)) \geq \left(1 - d\left(1 - \left(1 - \frac{1}{|\Omega|^w}\right)^{n-w+1}\right)\right) \cdot \log_2\left(|\mathcal{CK}| - |\mathcal{CK}| \cdot d\left(1 - \left(1 - \frac{1}{|\Omega|^w}\right)^{n-w+1}\right)\right) .$$

Proof. Let $\#_s$ denote the number of cookies $ck \in \mathcal{CK}$ such that $\#\mathrm{SUB}(ck) = s$. First note that

$$\Pr(\#\mathrm{SUB}(CK) = s) = \frac{\#_s}{|\mathcal{CK}|} .$$

Additionally,

$$\Pr(CK = ck \mid \#\mathrm{SUB}(CK) = s) = \begin{cases} \frac{1}{\#_s}, & \text{if } \#\mathrm{SUB}(CK) = s \ , \\ 0, & \text{otherwise} \ . \end{cases}$$

Substituting into the definition of conditional entropy, $H(CK \mid \#\mathrm{SUB}(CK))$

$$\begin{aligned} &= -\sum_{s \in \mathbb{N}} \Pr(\#\mathrm{SUB}(CK) = s) \\ &\quad \cdot \sum_{ck \in \mathcal{CK}} \Pr(CK = ck \mid \#\mathrm{SUB}(CK) = s) \log_2 \Pr(CK = ck \mid \#\mathrm{SUB}(CK) = s) \\ &= -\sum_{s \in \mathbb{N}} \frac{\#_s}{|\mathcal{CK}|} \sum_{ck \in \mathcal{CK} : \#\mathrm{SUB}(CK) = s} \frac{1}{\#_s} \log_2 \frac{1}{\#_s} \\ &= \frac{1}{|\mathcal{CK}|} \sum_{s \in \mathbb{N}} \#_s \log_2 \#_s \ . \end{aligned}$$

Let $\#_{\geq 1}$ denote the number of cookies $ck \in \mathcal{CK}$ such that $\#\mathrm{SUB}(ck) \geq 1$. Then

$$\begin{aligned} \Pr(\#\mathrm{SUB}(CK) \geq 1) = \Pr(\exists x \in \mathcal{D} : x \preceq ck) &= \frac{\#_{\geq 1}}{|\mathcal{CK}|} \quad \text{(by definition of } \#_{\geq 1}) \\ &\leq d\left(1 - \left(1 - \frac{1}{|\Omega|^w}\right)^{n-w+1}\right) \quad \text{(by Lemma 6.5.2)} \end{aligned}$$

Thus, the number of cookies with at least 1 substring in the dictionary is

$$\#_{\geq 1} \leq |\mathcal{CK}| \cdot d \left(1 - \left(1 - \frac{1}{|\Omega|^w}\right)^{n-w+1}\right) \ .$$

Consequently, the number of cookies with no substring in the dictionary is

$$\#_0 = |\mathcal{CK}| - \#_{\geq 1} \geq |\mathcal{CK}| - |\mathcal{CK}| d \left(1 - \left(1 - \frac{1}{|\Omega|^w}\right)^{n-w+1}\right) \ .$$

Finally,

$$\begin{aligned} H(CK \mid \#\mathrm{SUB}(CK)) &= \frac{1}{|\mathcal{CK}|} \sum_{s \in \mathbb{N}} \#_s \log_2 \#_s \geq \frac{1}{|\mathcal{CK}|} \#_0 \log_2 \#_0 \\ &\geq \left(1 - d \left(1 - \left(1 - \frac{1}{|\Omega|^w}\right)^{n-w+1}\right)\right) \\ &\quad \cdot \log_2 \left(|\mathcal{CK}| - |\mathcal{CK}| \cdot d \left(1 - \left(1 - \frac{1}{|\Omega|^w}\right)^{n-w+1}\right)\right) \square \end{aligned}$$

For example, if we have 16-byte cookies ($\mathcal{CK} = \{\texttt{0x00}, \ldots, \texttt{0xFF}\}^{16}$), and the dictionary $\mathcal{D}$ is a set of $d = 4096$ words of length $w = 4$ bytes, then

$$H(CK \mid \#\mathrm{SUB}(CK)) \geq 127.998395 \ .$$

Concluding our analysis of the information learned given to the adversary without any adversarially chosen prefix or suffix, we give a bound on the amount of entropy about the cookie given the length of the compressed cookie:

Lemma 6.5.4. Fix $\mathcal{D}$ with d words of length w over character set Ω. Denote the length of a cookie ck compressed with dictionary $\mathcal{D}$ by $\mathrm{COMPLEN}(ck) = |\mathrm{FD}_{\mathcal{D},w,\ell}.\mathrm{Comp}(ck)|$. Suppose CK is a uniform random variable on $\mathcal{CK}$. Then

$$\begin{aligned} &H(CK \mid \mathrm{COMPLEN}(CK)) \geq H(CK \mid \#\mathrm{SUB}(CK)) \\ &\geq \left(1 - d \left(1 - \left(1 - \frac{1}{|\Omega|^w}\right)^{n-w+1}\right)\right) \\ &\quad \cdot \log_2 \left(|\mathcal{CK}| - |\mathcal{CK}| \cdot d \left(1 - \left(1 - \frac{1}{|\Omega|^w}\right)^{n-w+1}\right)\right) \ . \end{aligned}$$

Lemma 6.5.4 follows from the data processing inequality and Lemma 6.5.3.

Probability bounds, prefix/suffix. Suppose CK is a uniform random variable on $\mathcal{CK} = \Omega^n$. We know that $H(CK) = n\log_2(|\Omega|)$. Trivially, $H(CK \mid CK_1) = (n-1)\log_2(|\Omega|)$, where CK_1 is the first character of CK. Similarly, $H(CK \mid CK_{1:a}) = (n-a)\log_2(|\Omega|)$ and finally $H(CK \mid CK_{1:a}, CK_{n-b:b}) = (n-a-b)\log_2(|\Omega|)$.

Consider the following CRIME-like attack on the beginning of the cookie. Let $\mathcal{D}$ be a dictionary with d words of length w over character set Ω. Let $ck \in \Omega^n$. Let $O(\cdot)$ be an oracle that, upon input a of length $w-m$, with $1 \leq m \leq w-1$, returns 1 if and only if $a \| ck_{1:m} \in \mathcal{D}$.

The CRIME-like attack works as follows:

1. For each $x \in \mathcal{D}$, query $x_{1:w-1}$ to the oracle. If a query for $x_{1:w-1}$ returns 1, then it is known that $ck_{1:1} \in Z_1 = \{z : x_{1:w-1} \| z \in \mathcal{D}\}$. If no query returns 1, then return $\emptyset$.

2. For $m = 2, \ldots, w-1$: For each $x \in \mathcal{D}$ such that $x_{w-m} \in Z_{m-1}$, query $x_{1:w-m}$ to the oracle. If a query for $x_{1:w-m}$ returns 1, then it is known that $ck_{1:m} \in Z_m = \{z_1 z_2 \ldots z_m : x_{1:w-m} \| z_1 z_2 \ldots z_m \in \mathcal{D}\}$. If no query returns 1, then return $Z_1, \ldots, Z_{m-1}$.

3. Return $Z_1, \ldots, Z_{w-1}$.

A corresponding attack on the suffix is obvious.

Let CRIMEpre(ck) denote the output obtained from running the above prefix CRIME attacks on ck, CRIMEsuf(ck) denote the output from the corresponding suffix attack. Let CRIME(ck) = (CRIMEpre(ck), CRIMEsuf(ck)).

Noting that in the best case the CRIME attack allows the attacker to learn the first $w-1$ and the last $w-1$ characters of the cookie, some trivial lower bounds are:

$$H(CK_{1:w-1} \mid \text{CRIME}(CK)) \geq 0$$
$$H(CK_{n-w+1:w-1} \mid \text{CRIME}(CK)) \geq 0$$

However, the CRIME attack provides no information about the remaining characters, so $I(CK_{1:w-1}, CK_{w:n-w+1}) = 0$ and $I(CK_{1:n-w+1}, CK_{n-w+1:w-1}) = 0$,

and thus $H(CK_{w:n-w+2} \mid \text{CRIME}(CK), \text{COMPLEN}(CK)) = H(CK_{w:n-w+2} \mid \text{COMPLEN}(CK))$.

Finally, we have that

$$\begin{aligned} &H(CK \mid \text{CRIME}(CK), \text{COMPLEN}(CK)) \\ &\quad \geq H(CK_{1:w-1} \mid \text{CRIMEpre}(CK)) + H(CK_{w:n-w+2} \mid \text{COMPLEN}(CK)) \\ &\qquad + H(CK_{n-w+1:w-1} \mid \text{CRIMEsuf}(CK)) \\ &\quad \geq 0 + H(CK_{w:n-w+2} \mid \text{COMPLEN}(CK)) + 0 \end{aligned}$$

and we can obtain a lower bound on $H(CK_{w:n-w} \mid \text{COMPLEN}(CK))$ using Lemma 6.5.4. □

6.5.3 Results on Fixed-Dictionary Technique

Table 6.1 shows the result of applying a fixed-dictionary based compression algorithm to the top 10 global websites of Alexa Top Sites. The 4000-byte dictionary was built from the most common 8-, 16-, and 32-character substrings of the pages. The compression algorithm was based in part on the Smaz [San09] algorithm and was adapted slightly from Figure 6.15, to allow for variable-length words to be matched. Specifically, when attempting to encode the substring at the current position at line 4 in Figure 6.15, we first try variable length words in order of decreasing length, checking to see if $w = 18$, then $w = 16$, then ..., then $w = 4$ characters can be found in the dictionary. This requires the encoding to include both index and length of the dictionary substring.

- To encode a dictionary word at index $0 \leq j < 4096$ of length $w = 2w'+4, 0 \leq w' \leq 7$, store 16 bits: `1` $\|$ [12-bit encoding of j] $\|$ [3-bit encoding of w']
- To encode 2 lower-ASCII characters $z_1 z_2$, store 16 bits: `00` $\|$ [7-bit encoding of z_1] $\|$ [7-bit encoding of z_2]
- To encode 1 byte z, store 16 bits: `01000000` $\|$ [8-bit encoding of z]

Source code of constructing a fixed-dictionary and fixed-dictionary experiment are in Appendix A.1 and A.2 respectively.

6.5.4 Discussion

The main drawback of the fixed dictionary mitigation technique is that in practice it achieves relatively poor—albeit non-zero—compression compared with adaptive compression techniques. However, it does not rely on application-dependent or heuristic techniques for separating secrets.

6.6 Summary

We introduced theoretical models to analyze compression-based side-channel attacks on high-value secrets embedded inside messages: the notions of cookie recovery (CR) security, random cookie indistinguishability (RCI), and chosen cookie indistinguishability (CCI). Each notion allows an attacker adaptive access to an oracle which encrypts chosen plaintexts alongside a target secret. We also characterize the relationship among the CCI and CR security notions, as well as an intermediate notion called *random cookie indistinguishability* (RCI) and the ER-IND-CPA notion of Kelley and Tamassia [KT14]. To some extent, the side channel exposed by compression is fundamentally unavoidable: if transmission of data is decreased, nothing can hide the fact that some redundancy existed in the plaintext. Hence, we focus our study on the ability of the attacker to learn specific "high value" secrets embedded in a plaintext, such as cookies or anti-CSRF tokens. In our models, we imagine there is a secret value ck, and the adversary can adaptively obtain encryptions

$$\mathrm{Enc}_k(m' \| ck \| m'') \tag{6.1}$$

for prefix m' and suffix m'' of its choice; the attacker's goal is to learn about ck.

The first mitigation technique we consider is that of *separating secrets*. During compression/encryption, an application-aware filter is applied to the plaintext to separate out any potential secret values from the data, the remaining plaintext is compressed, then the secrets and compressed plaintext are encrypted; after decryption, the inverse of the filter is used to reinsert the secret values in the decompressed plaintext. Assuming the filter fully separates out all secret values, we show that the separating secrets technique is able to achieve a strong notion of protection, which we call *chosen cookie indistinguishability* (CCI): the adversary cannot determine which of two cookies ck_0 and ck_1 of the adversary's choice was

encrypted with messages of the adversary's choice given ciphertexts as in (6.1).

The second mitigation technique we consider is the use of a *fixed-dictionary compression scheme*, where the dictionary used for compression does not adapt to the plaintext being compressed, but instead is preselected in advance based on the expected distribution of plaintext messages, for example including common English words like "the" and "and". We show that, if the secret values are sufficiently high entropy, then fixed-dictionary compression is able to achieve *cookie recovery* (CR) security: if the secret cookie is chosen uniformly at random, the adversary cannot recover the entire secret cookie even given an adaptive message attack as in (6.1). While cookie recovery security does not meet the "gold standard" of indistinguishability notions for encryption, it may be sufficient for some settings, for example protecting compressed HTTP traffic from CRIME and BREACH attacks that try to recover cookies and anti-CSRF tokens.

In the separating secrets technique, if the number of secrets extracted by the separating filter is relatively small, then the compressibility generally remains close to that of normal compression of the full plaintext. In the fixed-dictionary compression technique, compressibility suffers quite a bit compared to adaptive techniques on the full plaintext, although if the dictionary is constructed from a corpus of text similar to the plaintext, then some compression can be achieved.

Figure 6.16 summarizes experimental results comparing compression ratios for these two techniques on the HTML, CSS, and Javascript source code of the top 10 global websites as reported by Alexa Top Sites (`http://www.alexa.com/topsites`). On average, the compression ratio (uncompressed : compressed size) of gzip applied to the full source code was 5.42×; applying a separation filter that extracted all values following `value=` in the HTML source code yielded an average compression ratio of 5.20×; compression of each page using a fixed dictionary trained on all 10 pages yielded an average compression ratio of 1.55×.

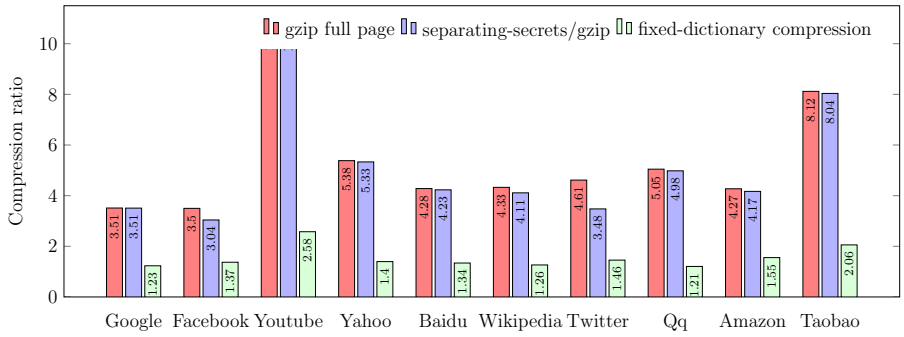

Figure 6.16: Compression ratios of full page compression versus mitigation techniques

Chapter 7

Conclusion and Future Works

Contents

We now summarize the thesis and discuss some possible research directions.

7.1 Summary

Typically, secure channels are constructed from an authenticated key exchange protocol which authenticates the communicating parties based on long-term public keys and establishes secret keys, and a secure data transmission layer which uses the secret keys to transmit data. In this research we model the partial leakage of long-term secret keys of key exchange protocol participants due to various side-channel attacks, and the partial leakage of plaintexts due to data compression prior to the encryption. We then constructed leakage-resilient key exchange protocols and encryption schemes satisfying our security models. The contributions of this research, together with the previous works on leakage-resilient symmetric-key encryption, can be used to build a stronger leakage-resilient secure channel suite for communication. Further, this research opens up new research directions, particularly pointing the research community towards the after-the-fact leakage of key exchange protocols, and compression-based plaintext leakage of symmetric-key encryption schemes.

The main contributions of this thesis are summarized as follows.

Chapter 3: New eCK**-secure protocol.** We contribute a new key exchange protocol construction and provide its security proof in the eCK model. This protocol does not use the so-called NAXOS trick, which has been used for many eCK-secure key exchange protocols.

Chapter 4: Long-term secret key leakage in the key exchange phase. This chapter contributes a new leakage security model: the continuous after-the-fact leakage in restricted-eCK (CAFL) model, which addresses continuous and after-the-fact leakage of long-term secret keys, while restricting the freshness condition, more than in the eCK or Moriyama-Okamoto model. Further, we construct a new generic protocol which is proven-secure in the CAFL model, using any suitable leakage-resilient public-key encryption scheme. Thus, contributions in this chapter provides a solution for continuous and after-the-fact leakage of long-term secret keys of key exchange protocol principals.

Chapter 5: Long-term secret key leakage in the key exchange phase (handshake layer). This chapter contributes a generic after-the-fact leakage security model for key exchange: the generic after-the-fact leakage eCK ((·)AFL-eCK) model, which can be instantiated as a bounded or continuous leakage variant. This model contains an eCK-style freshness condition, which is also appears in the eCK and Moriyama-Okamoto models. Thus, it releases the additional restrictions we enforced in Chapter 4 and its stronger than previously published models. We present a new generic protocol construction which is proven secure in the w(·)AFL-eCK model, using any suitable leakage-resilient public-key encryption scheme and any suitable leakage-resilient signature scheme (this protocol construction is slightly weaker in security than the desired protocol). Then, we give a new concrete construction of a key exchange protocol which is proven-secure in the continuous leakage variant of the (·)AFL-eCK model (CAFL-eCK model), using a leakage-resilient storage scheme and its refreshing protocol, and based on Diffie-Hellman key exchange (this protocol construction satisfies the desired security features). Thus, contributions in this chapter provide solutions for continuous/bounded and after-the-fact leakage of long-term secret keys of key exchange protocol principals, with more powerful eCK-style security compared to the model and the protocol presented in Chapter 4.

Chapter 6: Compression-based leakage attacks in the data transmission phase (record layer). This chapter contributes security notions to analyze compression-based leakage on symmetric-key encryption schemes. Then, we examine two possible mitigation techniques – separating-secrets from user inputs and fixed-dictionary compression – and prove their security in new security notions. Further, we report on the amount of compressibility that they can achieve for a real world example. Thus, this chapter provide provable secure countermeasures against compression-based leakage attacks on symmetric-key encryption schemes, and open up new research directions in the context of compression-based leakage attacks.

7.2 Future Directions

Leakage-Resilient Key Exchange

As future work in the context of leakage-resilient key exchange, it is worthwhile to investigate following directions:

- In order to achieve after-the-fact leakage resiliency in Chapter 5 using the split-state technique, the secret is encoded into two parts which contains two n-element vectors or one n-element vector and one $m \times n$-element array, and only allows leakage independently from each encoding. Because of this approach the actual exponentiation operation needs many element-wise exponentiations, and the computation cost is high. Therefore, it is worthwhile to investigate whether there are other techniques to achieve after-the-fact leakage resilience, rather than encoding the secret into parts. Thus, it will be possible to construct more efficient leakage-resilient key exchange protocols.

- In this research we only consider the partial leakage of long-term secrets due to side-channel attacks. Since the ephemeral secret keys are short-time values and are not involved in as many computations as long-term secrets, the possibility of continuous leakage of ephemeral secret keys, due to side-channel attacks such as timing, power analysis or EM emission based attacks may not be possible. Differently, (bounded) leakage attacks which reveal the memory of the system, particularly cold boot attacks, can leak ephemeral secret keys to the adversary fully or partially. The Moriyama-Okamoto

model and the BAFL-eCK model, address the cold boot attacks to some extent, where the cold boot attacks reveal either

  - the full long-term secret key,
  - the full ephemeral secret key,
  - the full ephemeral secret key and part of the long-term secret key

of a protocol principal. But there are few other situations which have not been covered by the previous models: the cold boot attacks reveal

  - the full long-term secret key and part of the ephemeral secret key,
  - part of the long-term secret key and part of the ephemeral secret key

of a protocol principal. Thus, it is worthwhile to investigate about partial leakage of the ephemeral secret keys.

- In this research we do not consider the partial leakage of session keys, either during the session key derivation phase or when it is used for encryption. Even though it is different from the traditional idea of secret key leakage, it is worthwhile to investigate the partial leakage of session keys, particularly as an important aspect for leakage-resilient key exchange.

- Currently, the constructions for achieving leakage-resilient primitives are based on discrete-logarithm-type assumptions. When quantum computers become available, such primitives may no longer be secure. Therefore, it is worthwhile to investigate about quantum-safe leakage-resilient public-key encryption schemes, key exchange protocols etc.

Compression-based Plaintext Leakage

As future work in the context of compression-based plaintext leakage, it is worthwhile to investigate following directions:

- As future work, it is worthwhile to investigate on further cryptographic study of compression, including the investigation of definitions that provide both cookie indistinguishability and some measure of message indistinguishability.

- Further, developing a suitable framework to separate secrets automatically from web pages when applying compression-encryption would be an interesting real-world application. Thus, web developers can use that framework

and achieve security against compression-based leakage attacks to some extent for known data field which contain secrets.

- Moreover, developing optimum fixed dictionary for web pages would be an another interesting direction for the real-world applications.

Appendix A

Source Code

Contents

A.1 Constructing a Fixed-Dictionary

```
1 import java.nio.file.*;
2 import java.util.*;
3 public class Freqs {
4         public static final int RECORD_SIZE = 2;
5         public static void main(String args[]) throws Exception {
6         if (args.length < 3) {
7                 System.err.println("Usage: java Freqs dictSize
                      wordLengths filenames > output.dict");
8                 System.err.println("- dictSize: the target size of
                      the output dictionary in bytes; for Spaz this
                      should be 4000");
9                 System.err.println("- wordLengths: comma-separated
                      list of word lengths to count frequencies of; e.
                      g., 8,15,20");
```

```
10                  System.err.println("- filenames: list of filenames
                        to build dictionary from");
11                  return;
12          }
13          int dictSize = Integer.parseInt(args[0]);
14          String wordlengths[] = args[1].split(",");
15          HashMap<String, Integer> worths = new HashMap<String,
                Integer>();
16          for (int f = 2; f < args.length; f++) {
17                  String filename = args[f];
18                  String src = new String(Files.readAllBytes(Paths.get
                        (filename)));
19                  for (String wordlength : wordlengths) {
20                          calcWorth(src, worths, Integer.parseInt(
                                wordlength));
21                  }
22          }
23          printBest(worths, dictSize);
24  }
25
26  static void calcWorth(String src, Map<String, Integer> freqs, int
        seqlen) {
27          for (int i = 0; i < src.length() - seqlen; i++) {
28                  String s = src.substring(i, i + seqlen);
29                  if (freqs.containsKey(s)) {
30                          freqs.put(s, freqs.get(s) + seqlen -
                                RECORD_SIZE);
31                  } else {
32                          freqs.put(s, seqlen - RECORD_SIZE);
33                          }
34          }
35  }
36
37  static void printBest(Map<String, Integer> freqs, int dictSize) {
38          Map<String, Integer> freqs_sorted = Freqs.sortByValue(freqs)
                ;
39          Vector<String> best = new Vector<String>(dictSize);
40          Iterator<Map.Entry<String, Integer>> it = freqs_sorted.
                entrySet().iterator();
41          int currDictSize = 0;
42          while (it.hasNext() && (currDictSize < dictSize)) {
43                  Map.Entry<String, Integer> pair = it.next();
44                  boolean useit = true;
```

```
45                  for (String v : best) {
46                          if (longestSubstr(pair.getKey(), v) > 5) {
47                                  useit = false;
48                                  break;
49                          }
50                  }
51                  if (useit) {
52                          best.add(pair.getKey());
53                          System.out.print(pair.getKey());
54                          currDictSize += pair.getKey().length();
55                  }
56          }
57          System.out.println();
58  }
59
60  public static <K, V extends Comparable<? super V>> Map<K, V>
        sortByValue(Map<K, V> map) {
61          List<Map.Entry<K, V>> list = new LinkedList<Map.Entry<K, V
                >>(map.entrySet());
62          Collections.sort(list, new Comparator<Map.Entry<K, V>>() {
63                  public int compare(Map.Entry<K, V> o1, Map.Entry<K,
                        V> o2) {
64                          return -(o1.getValue()).compareTo(o2.
                                getValue());
65                  }
66          });
67          Map<K, V> result = new LinkedHashMap<K, V>();
68          for (Map.Entry<K, V> entry : list) {
69                  result.put(entry.getKey(), entry.getValue());
70          }
71          return result;
72  }
73
74  public static int longestSubstr(String first, String second) {
75          if (first == null || second == null || first.length() == 0
                || second.length() == 0) {
76                  return 0;
77          }
78          int maxLen = 0;
79          int fl = first.length();
80          int sl = second.length();
81          int[][] table = new int[fl + 1][sl + 1];
82          for (int s = 0; s <= sl; s++)
```

```
83                  table[0][s] = 0;
84          for (int f = 0; f <= fl; f++)
85                  table[f][0] = 0;
86          for (int i = 1; i <= fl; i++) {
87                  for (int j = 1; j <= sl; j++) {
88                          if (first.charAt(i - 1) == second.charAt(j -
                                1)) {
89                                  if (i == 1 || j == 1) {
90                                          table[i][j] = 1;
91                                  } else {
92                                          table[i][j] = table[i - 1][j
                                                - 1] + 1;
93                                  }
94                                  if (table[i][j] > maxLen) {
95                                          maxLen = table[i][j];
96                                  }
97                          }
98                  }
99          }
100                 return maxLen;
101         }
102 }
```

A.2 Fixed-Dictionary Experiment

```
1  import java.nio.file.*;
2  public class Spaz2 {
3          public static final String dictionary = "dictionary";
4          public static String decode(byte[] c, String dictionary)
               throws Exception {
5                   byte bytes[] = new byte[8*c.length];
6                   int bindex = 0;
7                   int cindex = 0;
8                   while (cindex < c.length) {
9                           byte b = c[cindex];
10                          if (b >> 6 == 0) {
11                                  if (cindex + 1 >= c.length) {
12                                          throw new Exception("Need 2
                                                byte record");
13                                  }
14                          byte a = c[cindex];
15                          a &= 0x3F;
16                          a <<= 1;
```

```
17                          byte a2 = c[cindex+1];
18                          a2 >>>= 7;
19                          a2 &= 0x01;
20                          a |= a2;
21                          bytes[bindex] = a;
22                          a = c[cindex+1];
23                          a &= 0x7F;
24                          bytes[bindex+1] = a;
25                          cindex += 2;
26                          bindex += 2;
27                          } else if ((b & 0x80) > 0) {
28                                  if (cindex + 1 >= c.length) {
29                                          throw new Exception("Need 2
                                                byte record");
30                                  }
31                          int j = c[cindex+1] & 0x07;
32                          int len = 2 * j + 4;
33                          int index = c[cindex] & 0x7F;
34                          index <<= 5;
35                          int index2 = c[cindex+1] & 0xF8;
36                          index2 >>= 3;
37                          index2 &= 0x1F;
38                          index |= index2;
39                          String dictlookup = dictionary.substring(
                                index, index + len);
40                          byte dictlookupbytes[] = dictlookup.getBytes
                                ("UTF-8");
41                          System.arraycopy(dictlookupbytes, 0, bytes,
                                bindex, dictlookupbytes.length);
42                          cindex += 2;
43                          bindex += dictlookupbytes.length;
44                          } else if ((b & 0x40) > 0) {
45                                  if (cindex + 1 >= c.length) {
46                                          throw new Exception("Need 2
                                                byte record");
47                                  }
48                                  bytes[bindex] = c[cindex+1];
49                                  cindex += 2;
50                                  bindex += 1;
51                          } else {
52                          throw new Exception("Unknown record type");
53                  }
54          }
```

```
55          return new String(bytes, 0, bindex, "UTF-8");
56  }
57
58  static boolean doit(byte[] c, int cindex, String dictionary, byte[]
        b, int bindex, int seqlen) {
59          if (cindex + seqlen <= c.length) {
60                  int dictindex = dictionary.indexOf(new String(c,
                        cindex, seqlen));
61                  if (dictindex >= 0) {
62                          int constructed = 0x00008000;
63                          constructed |= dictindex << 3;
64                          constructed |= (seqlen-4) / 2;
65                          b[bindex] = (byte) ((constructed >>> 8) & 0
                                xFF);
66                          b[bindex+1] = (byte) (constructed & 0xFF);
67                          return true;
68                  }
69          }
70          return false;
71  }
72
73  public static byte[] encode(String s, String dictionary) throws
        Exception {
74          byte[] b = new byte[2 * s.length()];
75          byte[] c = s.getBytes("UTF-8");
76          int cindex = 0;
77          int bindex = 0;
78          int freqs[] = new int[30];
79          for (int i = 0; i < 30; i++) { freqs[i] = 0; }
80          while (cindex < c.length) {
81                  boolean foundone = false;
82                  for (int j = 7; j >= 0; j--) {
83                          int seqlen = 2 * j + 4;
84                          if (doit(c, cindex, dictionary, b, bindex,
                                seqlen)) {
85                                  cindex += seqlen;
86                                  bindex += 2;
87                                  freqs[seqlen]++;
88                                  foundone = true;
89                                  break;
90                          }
91                  }
92                  if (foundone) continue;
```

```
93                  if ((cindex + 2 <= c.length) && ((c[cindex] >= 0x00)
                        && (c[cindex] < 0x7F)) && ((c[cindex+1] >= 0x00
                        ) && (c[cindex+1] < 0x7F))){
94                          b[bindex] = (byte) (c[cindex] >> 1);
95                          b[bindex] &= 0x3F;
96                          b[bindex+1] = (byte) c[cindex+1];
97                          b[bindex+1] &= 0x7F;
98                          b[bindex+1] |= (byte) (c[cindex] << 7);
99                          bindex += 2;
100                         cindex += 2;
101                         freqs[2]++;
102                 } else {
103                         b[bindex] = (byte) 0x40;
104                         b[bindex+1] = (byte) c[cindex];
105                         bindex += 2;
106                         cindex += 1;
107                         freqs[1]++;
108                 }
109         }
110         byte[] r = new byte[bindex];
111         System.arraycopy(b, 0, r, 0, bindex);
112         System.out.printf("freqs: ");
113         for (int i = 0; i < 30; i++) {
114                 if (freqs[i] > 0) {
115                         System.out.printf("%dx%d, ", freqs[i], i);
116                 }
117         }
118         System.out.println();
119         return r;
120 }
121
122 public static void main(String args[]) {
123         if (args.length != 2) {
124                 System.err.println("Usage: Spaz dictionary input");
125                 return;
126         }
127         String dictionaryFilename = args[0];
128         String inputFilename = args[1];
129         try {
130                 String dictionary = new String(Files.readAllBytes(
                        Paths.get(dictionaryFilename)));
131                 String input = new String(Files.readAllBytes(Paths.
                        get(inputFilename)));
```

```
132                 if (dictionary.length() > 4096) {
133                         throw new Exception("Dictionary is too long.
                            ");
134                 }
135                 byte[] b = Spaz2.encode(input, dictionary);
136                 String output = Spaz2.decode(b, dictionary);
137                 int inputLength = input.getBytes("UTF-8").length;
138                 int outputLength = b.length;
139                 double compression = ((double) inputLength) / ((
                        double) outputLength);
140                 System.out.printf("Input length: %d; output length:
                        %d; compression: %.3fx\n", inputLength,
                        outputLength, compression);
141                 if (!input.equals(output)) {
142                         throw new Exception("Strings do not match.")
                                ;
143                 }
144                 } catch (Exception e) {
145                         e.printStackTrace();
146                 }
147         }
148 }
```

A.3 Separating-Secrets Mitigation Technique

```
1  import java.io.*;
2  import java.util.regex.Pattern;
3  import java.util.regex.Matcher;
4  import java.util.zip.GZIPOutputStream;
5  import java.util.zip.GZIPInputStream;
6  import java.nio.file.*;
7  public class exp{
8          /* method of separating secrets; separate contents assigned
               to the ``value" attribute*/
9          public static void SepSec(String original_file){
10                 try{
11                         FileReader filereader = new FileReader(
                               original_file);
12                         BufferedReader in = new BufferedReader(
                               filereader);
13                         FileWriter filewriter_ns = new FileWriter(
                               original_file+"_ns");
```

```
14                          BufferedWriter bfr_ns = new BufferedWriter(
                                filewriter_ns);
15                          FileWriter filewriter_s = new FileWriter(
                                original_file+"_s");
16                          BufferedWriter bfr_s = new BufferedWriter(
                                filewriter_s);
17                          Pattern pattern = Pattern.compile("value\\s
                                *=\\s*\"([^\"]*)\"|value\\s*=\\s
                                *\'([^\']*)\'|value\\s*=\\s*[^\\s]*");
18                          Matcher p_matcher;
19                          String secret, nonsecret;
20                          String line = in.readLine();
21                          while (line != null){
22                                  nonsecret = line.replaceAll("value\\
                                        s*=\\s*\"([^\"]*)\"|value\\s*=\\
                                        s*\'([^\']*)\'|value\\s*=\\s
                                        *[^\\s]*", "FILTERED-OUT");
23                                  bfr_ns.write(nonsecret);
24                                  bfr_ns.newLine();
25                                  p_matcher = pattern.matcher(line);
26                                  while (p_matcher.find()) {
27                                          secret = p_matcher.group();
28                                                  bfr_s.write(secret);
29                                                  bfr_s.newLine();
30                                  }
31                                  line = in.readLine();
32                          }
33                          in.close();
34                          bfr_s.close();
35                          bfr_ns.close();
36                  }
37          catch(Exception e){
38                  System.out.print(e);
39          }
40  }
41
42  /*gzip compression; modified the code of Byron Kiourtzoglou (http://
        examples.javacodegeeks.com/core-java/io/fileinputstream/compress
        -a-file-in-gzip-format-in-java/)*/
43          public static void gzipFile(String source_filepath, String
                destinaton_zip_filepath) {
44                  byte[] buffer = new byte[1024];
45                  try {
```

```
46                          FileOutputStream fileOutputStream =new
                                FileOutputStream(destinaton_zip_filepath
                                );
47                          GZIPOutputStream gzipOuputStream = new
                                GZIPOutputStream(fileOutputStream);
48                          FileInputStream fileInput = new
                                FileInputStream(source_filepath);
49                          int bytes_read;
50                          while ((bytes_read = fileInput.read(buffer))
                                 > 0){
51                                  gzipOuputStream.write(buffer, 0,
                                        bytes_read);
52                          }
53                          fileInput.close();
54                          gzipOuputStream.finish();
55                          gzipOuputStream.close();
56                  }
57                  catch (Exception e){
58                          System.out.print(e);
59                  }
60          }
61
62  /*gunzip decompression; modified the code of Nikos Maravitsas (http
        ://examples.javacodegeeks.com/core-java/io/fileinputstream/
        decompress-a-gzip-file-in-java-example/)*/
63          public static void unGunzipFile(String compressedFile,
                String decompressedFile){
64                  byte[] buffer = new byte[1024];
65                  try{
66                          FileInputStream fileIn = new FileInputStream
                                (compressedFile);
67                  GZIPInputStream gZIPInputStream = new
                        GZIPInputStream(fileIn);
68                          FileOutputStream fileOutputStream = new
                                FileOutputStream(decompressedFile);
69                  int bytes_read;
70                  while ((bytes_read = gZIPInputStream.read(buffer)) >
                         0) {
71                          fileOutputStream.write(buffer, 0, bytes_read
                                );
72                  }
73                      gZIPInputStream.close();
74                  fileOutputStream.close();
```

```
75                  }
76                  catch (IOException ex){
77                  ex.printStackTrace();
78              }
79          }
80
81
82  /* method of recovering the original file from secrets and non-
        secrets*/
83          public static void MergeSec(String secret_file, String
                non_secret_file){
84                  try{
85                          FileWriter filewriter = new FileWriter(
                                non_secret_file+"_recover.htm");
86                          BufferedWriter out = new BufferedWriter(
                                filewriter);
87                          FileReader filereader_ns = new FileReader(
                                non_secret_file);
88                          BufferedReader bfr_ns = new BufferedReader(
                                filereader_ns);
89                          FileReader filereader_s = new FileReader(
                                secret_file);
90                          BufferedReader bfr_s = new BufferedReader(
                                filereader_s);
91                          Pattern pattern = Pattern.compile("FILTERED-
                               OUT");
92                          Matcher p_matcher;
93                          String line, secret;
94                          String recovery;
95                          StringBuffer sb;
96                          line = bfr_ns.readLine();
97                          while (line != null){
98                                  sb = new StringBuffer();
99                                  p_matcher = pattern.matcher(line);
100                                 while (p_matcher.find()) {
101                                         String text = p_matcher.
                                                group(0);
102                                         secret = bfr_s.readLine();
103                                         p_matcher.appendReplacement(
                                                sb, p_matcher.
                                                quoteReplacement(secret)
                                                );
104                                 }
```

```
105                             p_matcher.appendTail(sb);
106                             recovery = sb.toString();
107                             out.write(recovery);
108                             out.newLine();
109                             line = bfr_ns.readLine();
110                     }
111                     out.close();
112                     bfr_s.close();
113                     bfr_ns.close();
114             }
115             catch(Exception e){
116                     System.out.print(e);
117             }
118     }
```

A.4 Separating-Secrets Experiment

```
1       /*experiment*/
2       public static void main(String args[]){
3               try{
4                       float original_size, compressed_size,
                            mitigated_size, secret_size,
                            non_secret_size, avg1 = 0, avg2 = 0;
5                       FileReader filenames = new FileReader("names
                            .txt");
6                       BufferedReader f_names = new BufferedReader(
                            filenames);
7                       FileWriter w = new FileWriter("Report.text")
                            ;
8                       BufferedWriter out = new BufferedWriter(w);
9                       String line = f_names.readLine();
10                      while (line != null){
11                              String temp = new String(Files.
                                    readAllBytes(Paths.get(line)));
12                              original_size = temp.getBytes("UTF-8
                                    ").length;
13                              gzipFile(line, line+".gzip");
14                              compressed_size = Files.readAllBytes
                                    (Paths.get(line+".gzip")).length
                                    ;
15                              SepSec(line);
16                              secret_size = Files.readAllBytes(
                                    Paths.get(line+"_s")).length;
```

```
17                  gzipFile(line+"_ns", line+"_ns.gzip");
18                  non_secret_size = Files.readAllBytes(Paths.get(line+"_ns.gzip")).length;
19                  mitigated_size = non_secret_size + secret_size;
20                  unGunzipFile(line+"_ns.gzip", line+"2_ns");
21                  MergeSec(line+"_s", line+"2_ns");
22                  out.write("File Name : "+line);
23                  out.newLine();
24                  out.write("--------------------------------");
25                  out.newLine();
26                  out.write("Size of the Original File : "+original_size+" bytes");
27                  out.newLine();
28                  out.write("Size of the Compressed File : "+compressed_size+" bytes");
29                  out.newLine();
30                  out.write("Compression : "+original_size/compressed_size);
31                  avg1 += original_size/compressed_size;
32                  out.newLine();
33                  out.write("Extracter and Compressed: "+non_secret_size+" bytes");
34                  out.newLine();
35                  out.write("Size of the Secrets : "+secret_size+" bytes");
36                  out.newLine();
37                  out.write("Total size the mitigation : "+mitigated_size+" bytes");
38                  out.newLine();
39                  out.write("Compression : "+original_size/mitigated_size);
40                  avg2 += original_size/mitigated_size;
41                  out.newLine();
42                  out.write("
```

```
                                        ");
43                                      out.newLine();
44                                      line = f_names.readLine();
45                              }
46                              out.write("Avg Original Compression : "+avg1
                                    /10);
47                              out.newLine();
48                              out.write("Avg mitigated Compression : "+
                                    avg2/10);
49                              out.close();
50                      }
51                      catch (Exception e){
52                              System.out.print(e);
53                      }
54              }
55              }
```

Bibliography

[ABR01] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle diffie-hellman assumptions and an analysis of DHIES. In *Topics in Cryptology - CT-RSA 2001, The Cryptographer's Track at RSA Conference 2001, San Francisco, CA, USA, April 8-12, 2001, Proceedings*, pages 143–158, 2001.

[ABS14] Janaka Alawatugoda, Colin Boyd, and Douglas Stebila. Continuous after-the-fact leakage-resilient key exchange. In *Information Security and Privacy - 19th Australasian Conference, ACISP 2014, Wollongong, NSW, Australia, July 7-9, 2014. Proceedings*, pages 258–273, 2014.

[ACF09] Michel Abdalla, Dario Catalano, and Dario Fiore. Verifiable random functions from identity-based key encapsulation. In *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, pages 554–571, 2009.

[ADW09] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In *CRYPTO*, pages 36–54, 2009.

[AGV09] Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *Theory of Cryptology Conference*, pages 474–495, 2009.

[AJR11] J. Alawatugoda, D. Jayasinghe, and R. Ragel. Countermeasures against Bernstein's remote cache timing attack. In *6th IEEE Inter-*

national Conference on Industrial and Information Systems (ICIIS), pages 43 –48, Aug. 2011.

[ASB14] Janaka Alawatugoda, Douglas Stebila, and Colin Boyd. Modelling after-the-fact leakage for key exchange. In *9th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '14, Kyoto, Japan - June 03 - 06, 2014*, pages 207–216, 2014.

[BCNP09] Colin Boyd, Yvonne Cliff, Juan Manuel González Nieto, and Kenneth G. Paterson. One-round key exchange in the standard model. *International Journal of Advanced Computer Technology*, pages 181–199, 2009.

[BDPR98] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In *CRYPTO*, pages 26–45, 1998.

[Ber05] Daniel J. Bernstein. Cache-timing attacks on AES. Technical report, 2005. http://cr.yp.to/antiforgery/cachetiming-20050414.pdf.

[BKKV10] Zvika Brakerski, Yael Tauman Kalai, Jonathan Katz, and Vinod Vaikuntanathan. Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage. *IACR Cryptology ePrint Archive*, Report 2010/278, 2010.

[BMP00] Victor Boyko, Philip MacKenzie, and Sarvar Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In *Proceedings of the 19th international conference on Theory and application of cryptographic techniques*, EUROCRYPT'00, pages 156–171, Berlin, Heidelberg, 2000. Springer-Verlag.

[BPT14] Mike Belshe, Roberto Peon, and Martin Thomson. Hypertext Transfer Protocol version 2, November 2014. Internet-Draft. `http://tools.ietf.org/html/draft-ietf-httpbis-http2-16`.

[BR93] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *CRYPTO*, pages 232–249, 1993.

[BR95] Mihir Bellare and Phillip Rogaway. Provably secure session key distribution - the three party case. pages 57–66. ACM Press, 1995.

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, pages 409–426, 2006.

[Cas00] J. L. Casti. *Five more golden rules : knots, codes, chaos, and other great theories of 20th century mathematics / John L. Casti*. Wiley New York ; Brisbane, 2000.

[CBHM04] Kim-Kwang Raymond Choo, Colin Boyd, Yvonne Hitchcock, and Greg Maitland. On session identifiers in provably secure protocols: The bellare-rogaway three-party key distribution protocol revisited. In *Security in Communication Networks, 4th International Conference, SCN 2004, Amalfi, Italy, September 8-10, 2004, Revised Selected Papers*, pages 351–366, 2004.

[CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *EUROCRYPT*, pages 453–474, 2001.

[Cre11] Cas Cremers. Examining indistinguishability-based security models for key exchange protocols: the case of ck, ck-hmqv, and eck. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2011, Hong Kong, China, March 22-24, 2011*, pages 80–91, 2011.

[CS02] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *EUROCRYPT*, pages 45–64. Springer, 2002.

[DF11] Stefan Dziembowski and Sebastian Faust. Leakage-resilient cryptography from the inner-product extractor. In *ASIACRYPT*, pages 702–721, 2011.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, pages 644 – 654, 1976.

[DHLAW10] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Efficient public-key cryptography in the presence of key leakage. In *ASIACRYPT*, pages 613–631, 2010.

[DKL09] Yevgeniy Dodis, Yael Tauman Kalai, and Shachar Lovett. On cryptography with auxiliary input. In *STOC*, pages 621–630, 2009.

[DP08] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *IEEE Symposium on Foundations of Computer Science*, pages 293–302, 2008.

[DR08] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176.

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *EUROCRYPT*, pages 523–540, 2004.

[DvOW92] Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *Des. Codes Cryptography*, 2(2):107–125, 1992.

[FKPR09] Sebastian Faust, Eike Kiltz, Krzysztof Pietrzak, and Guy N. Rothblum. Leakage-resilient signatures. *IACR Cryptology ePrint Archive*, Report 2009/282, 2009.

[FR14] R. Fielding and J. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230 (Proposed Standard), June 2014.

[FSU09] Atsushi Fujioka, Koutarou Suzuki, and Berkant Ustaoglu. Utilizing postponed ephemeral and pseudo-static keys in tripartite and identity-based key agreement protocols. *IACR Cryptology ePrint Archive*, 2009:423, 2009.

[GHP13] Yoel Gluck, Neal Harris, and Angelo Prado. SSL, gone in 30 seconds: A BREACH beyond CRIME. In *Black Hat USA 2013*, August 2013.

[HL11] Shai Halevi and Huijia Lin. After-the-fact leakage in public-key encryption. In *Theory of Cryptology Conference*, pages 107–124, 2011.

[HMF07] Michael Hutter, Stefan Mangard, and Martin Feldhofer. Power and EM attacks on passive 13.56MHz RFID devices. In *CHES*, pages 320–333, 2007.

[HR07] I. Herath and R.G. Ragel. Side channel attacks: Measures and countermeasures. In *14th Annual Conference of the IET Sri Lanka Network*, 2007.

[Jab96] David P. Jablon. Strong password-only authenticated key exchange. *SIGCOMM Comput. Commun. Rev.*, 26(5):5–26, October 1996.

[Kel02] John Kelsey. Compression and information leakage of plaintext. In Joan Daemen and Vincent Rijmen, editors, *FSE 2002*, volume 2365 of *LNCS*, pages 263–276. Springer, February 2002.

[KHJ+10] Demijan Klinc, Carmit Hazay, Ashish Jagmohan, Hugo Krawczyk, , and Tal Rabin. On compression of data encrypted with block ciphers. Cryptology ePrint Archive, Report 2010/477, 2010. `http://eprint.iacr.org/2010/477`.

[KL07] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. pages 104–113. Springer-Verlag, 1996.

[KPSY09] Eike Kiltz, Krzysztof Pietrzak, Martijn Stam, and Moti Yung. A new randomness extraction paradigm for hybrid encryption. In *EUROCRYPT*, pages 590–609, 2009.

[KR01] Joe Kilian and Phillip Rogaway. How to protect DES against exhaustive key search (an analysis of DESX). *J. Cryptology*, 14(1):17–35, 2001.

[Kra05] Hugo Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In *CRYPTO*, pages 546–566, 2005.

[Kra08] Hugo Krawczyk. On extract-then-expand key derivation functions and an HMAC-based KDF. http://webee.technion.ac.il/ hugo/kdf/kdf.pdf, 2008.

[KT14] James Kelley and Roberto Tamassia. Secure compression: Theory & practice. Cryptology ePrint Archive, Report 2014/113, 2014. `http://eprint.iacr.org/2014/113`.

[KV09] Jonathan Katz and Vinod Vaikuntanathan. Signature schemes with bounded leakage resilience. In *ASIACRYPT*, pages 703–720, 2009.

[LLM07] Brian LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In *ProvSec*, pages 1–16, 2007.

[LMQ+98] Laurie Law, Alfred Menezes, Minghua Qu, Jerry Solinas, and Scott Vanstone. An efficient protocol for authenticated key agreement. Technical report, Designs, Codes and Cryptography, 1998.

[McC91] John R. McCumber. Information systems security : A comprehensive model. In *Proceedings of the 14th National Computer Security Conference*, October 1991.

[MDS02] T.S. Messerges, E.A. Dabbish, and R.H. Sloan. Examining smartcard security under the threat of power analysis attacks. *IEEE Transactions on Computers*, pages 541–552, 2002.

[MO09] Daisuke Moriyama and Tatsuaki Okamoto. An eck-secure authenticated key exchange protocol without random oracles. In *Provable Security, Third International Conference, ProvSec 2009, Guangzhou, China, November 11-13, 2009. Proceedings*, pages 154–167, 2009.

[MO11] Daisuke Moriyama and Tatsuaki Okamoto. Leakage resilient eCK-secure key exchange protocol without random oracles. In *ASIACCS*, pages 441–447, 2011.

[MR04] Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In *Theory of Cryptology Conference*, pages 278–296, 2004.

[MTVY11] Tal Malkin, Isamu Teranishi, Yevgeniy Vahlis, and Moti Yung. Signatures resilient to continual leakage on memory and computation. In *Theory of Cryptology Conference*, pages 89–106, 2011.

[NS09] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In *CRYPTO*, pages 18–35. 2009.

[Pie09a] Krzysztof Pietrzak. A leakage-resilient mode of operation. In *Proceedings of the 28th Annual International Conference on Advances in Cryptology: the Theory and Applications of Cryptographic Techniques*, EUROCRYPT '09, pages 462–482, Berlin, Heidelberg, 2009. Springer-Verlag.

[Pie09b] Krzysztof Pietrzak. A leakage-resilient mode of operation. In *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, pages 462–482, 2009.

[Pik80] J. Pike. Text compression using a 4 bit coding scheme. *The Computer Journal*, 24(4):324–330, September 1980.

[PR15] Roberto Peon and H Ruellan. HPACK-Header compression for HTTP/2, May 2015. `http://http2.github.io/http2-spec/compression.html`.

[RD12] Juliano Rizzo and Thai Duong. The CRIME attack, 2012. Presented at ekoparty '12. `http://goo.gl/mlw1X1`.

[San09] Salvatore Sanfilippo. Smaz: Small strings compression library, April 2009. `https://github.com/antirez/smaz`.

[Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. *IACR Cryptology ePrint Archive*, Report 2004/332, 2004.

[SPY13] François-Xavier Standaert, Olivier Pereira, and Yu Yu. Leakage-resilient symmetric cryptography under empirically verifiable assumptions. In *Advances in Cryptology - CRYPTO 2013 - 33rd*

Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I, pages 335–352, 2013.

[The09] The Chromium Projects. SPDY, 2009. `http://dev.chromium.org/spdy`.

[Too15] Mohsen Toorani. On continuous after-the-fact leakage-resilient key exchange. In *Proceedings of the Second Workshop on Cryptography and Security in Computing Systems, CS2@HiPEAC 2015, Amsterdam, Netherlands, January 19-21, 2015*, 2015.

[Tru15] Trustworthy Internet Movement. SSL Pulse, June 2015. `https://www.trustworthyinternet.org/ssl-pulse/`.

[Yan13] Zheng Yang. Efficient eck-secure authenticated key exchange protocols in the standard model. In *Information and Communications Security - 15th International Conference, ICICS 2013, Beijing, China, November 20-22, 2013. Proceedings*, pages 185–193, 2013.

[YMSW13] Guomin Yang, Yi Mu, Willy Susilo, and Duncan S. Wong. Leakage resilient authenticated key exchange secure in the auxiliary input model. In *ISPEC*, pages 204–217, 2013.